Netherlands organization for applied scientific research

TNO-report

TNO Physics and Electronics Laboratory

P.O. Box 96864
2509 JG The Hague
Oude Waalsdorperweg 63
The Hague, The Netherlands

Fax +31 70 328 09 61
Phone +31 70 326 42 21

report no. FEL-90-B131

copy no. 8

title

Garbled Text String Recognition with a Spatio-temporal Pattern Recognition Neural Network

90 1716

author:

P.P. Meiler

classification

title	:	Unclassified
abstract	:	Unclassified
report	:	Unclassified
appendix A	:	Unclassified

no. of copies	:	46
no. of pages	:	67 (incl. titlepage and appendix, excl. distr. list and RDP)
appendices	:	1
date	:	June 1990

All information which is classified according to Dutch regulations shall be treated by the recipient in the same way as classified information of corresponding value in his own country. No part of this information will be disclosed to any party.

report no.	:	FEL-90-B131
title	:	Garbled text string recognition with a spatio-temporal pattern recognition Neural Network
author	:	P.P. Meiler
institute	:	TNO Physics and Electronics Laboratory
date	:	June 1990
no. in pow '90	:	708

ABSTRACT (UNCLASSIFIED)

The purpose of this project is to show that neural networks can be used to recognize garbled words and/or parts of sentences in a **real-world** application. TNO-FEL has studied and built an application that recognizes the names of ships. These names may be garbled by transmission or typing errors, and synonyms or corruptions may be used. The SPR network emphasizes the character-sequence relationships within words. SPR is proof against missing, extra or interchanged (pairs of) characters. A learning strategy was developed and implemented. Several measurements were performed.

A-1

rapport no.	:	FEL-90-B131
titel	:	Verminkte tekst string herkenning met een spatio-temporeel patroon herkennings Neuraal Netwerk
auteur	:	P.P. Meiler
instituut	:	Fysisch en Elektronisch Laboratorium TNO
datum	:	Juni 1990
no. in iwp '90	:	708

SAMENVATTING (ONGERUBRICEERD)

Het doel van dit project is om aan te tonen dat neurale netwerken gebruikt kunnen worden om, in een **real-world** toepassing, verminkte woorden en/of delen van zinnen te herkennen. Er is een applicatie bestudeerd en gebouwd die in staat is de namen van schepen te herkennen. Deze namen kunnen verminkt zijn door transmissie- en/of typefouten. Ook kan een synoniem gebruikt zijn i.p.v. de officiële naam. Het SPR netwerk legt de nadruk op de karakter-volgorde relaties in een woord. SPR is bestand tegen weggevallen, extra tussengevoegde en verwisselde karakters. Er is een leerstrategie ontwikkeld en geïmplementeerd. De performance van het netwerk is d.m.v. een aantal metingen geëvalueerd.

INDEX

LIST OF FIGURES

1 INTRODUCTION

The Physics and Electronics Laboratory TNO (TNO-FEL) focuses primarily on observation, communication, information processing and operational research. To support the fast data processing usually required in sensory systems research, one of the research topics in Division 2 (System Development and Information Technology) comprises Parallel Processing. The major application areas covered by the Parallel Processing Group are:

- Radar data processing
- Real-time computer generated imagery (CGI-systems)
- 3-D image analysis, processing and visualization (voxel processing)
- Using **neural networks** in vision systems, multi sensor integration and other kinds of data-processing

Within the field of Parallel Processing, neural networks are becoming more and more popular. Because TNO is an Organization for **Applied** Scientific Research, our goal is not only to gather theoretical knowledge about neural network technology, but also to gain experience with the application of this knowledge. Recognizing text strings with a neural network was selected as a pilot project to gain some of that practical experience. This report presents the results of that project. Another pilot project is radar pulse classification with a BSB network [Wezenbeek 1990].

A text string recognition system can be very useful. Two examples of the usefulness of an automatic text string recognition system will be given.

1.1 Recognizing the names of ships

Consider a single channel radio broadcast net. A number of units exchange messages using this net. The messages may be sent using a code (e.g. ASCII). Each unit receives **all** messages that are sent by any of the other units. A unit could be a ship in a group of navy ships.

A message is recognized using the information in the header of the message. The header contains, among other data, the name of the addressee. This name (which is a text string) can then be used to recognize the message.

An automatic name recognition system which is capable to perform this task must be able to deal with different kinds of errors. A list of possible error sources:

- Spelling errors
- Typing errors
- Transmission errors.

1.2 Recognizing the names of cities

With Optical Character Recognition (OCR) technology it is possible to read the destination address (e.g. the name of a city) of a letter. This process is not perfect. Characters may be misinterpreted (e.g. a 'Q' may be read as an 'O' or vice-versa), or they may be totally unreadable (because of ink spots etc.). The sender of the letter may also make a spelling error.

Each country has a fixed set of cities. An automatic name recognition system finds the name of the city that matches best with the address on the letter, even if there are missing characters, etc.

1.3 A solution

Building an automatic name recognition system using conventional programming languages is very difficult. AI technology (e.g. Expert systems or Logic Programming) is more suitable. TNO-FEL is currently investigating the feasibility of such a system.

Due to its characteristics the problem was also chosen to be solved using "Neural Network Technology", as part of the research on parallel processing at TNO-FEL.

The name recognition problem was chosen because it is a real world problem that is large enough to have practical relevance and small enough to be manageable. The research was limited to the recognition of names (text strings). Operational requirements and constraints have not been taken into account.

The "Neural Network Technology" solution was implemented as a Spatio-temporal Pattern Recognition (SPR) neural network because the SPR concept fits very well with the fact that words consist of sequences of characters.

1.4 Outline of the rest of this report

The next section, "NEURAL NETWORK TECHNOLOGY", presents an elementary introduction to the theory and practice of neural network technology. The section "SPR NETWORK" will discuss the architecture and implementation of the SPR network, while the section "WEIGHT VECTORS" explains the learning rule that is used to train the SPR network (i.e. to set the weight vectors). The section "RESULTS" presents and explains the results of two measurements that were performed with the SPR network. The conclusions are given in the section "CONCLUSIONS". The literature that is referenced in this report is listed in the section "REFERENCES". Other relevant literature is listed in the section "LITERATURE". An Index and a List of Figures are provided at the start of this report. A visualization of the time varying activation levels of the PE's of the network is presented in appendix A.

2 NEURAL NETWORK TECHNOLOGY

Neural networks are massively parallel information processing systems. The structure and behaviour of a neural network is in some respects similar to a biological nervous system. A neural network consists of many, yet very simple, processing elements. A processing element will be referred to as **PE**. According to the biological model a PE can be compared with a neuron. The PE's of a neural network are, often very densely, connected to each other. A weight is associated with each connection between two PE's to determine its strength. Different neural network architectures may use different connection patterns or topologies (figure 1).

Instead of executing a program sequentially, as in classical von Neumann computers, all PE's of a neural network are active in parallel to explore many competing hypotheses at the same time. There is no central supervising unit in a neural network. Each PE processes its input data according to a predefined processing equation, independently of the other PE's. In fact, a neural network does not execute a program at all. The behaviour of a neural network is not programmed, it is learned using examples. For more information see [Grossberg 1988], [Lippmann 1987] and [McClelland & Rumelhart 1986].

A neural network is characterized by its paradigm. A neural network paradigm consists of the description of the following items:

- Topology of the network
- Processing equation describing the dynamics of a PE
- Network activation procedure
- Network learning procedure
- Data representation

Different neural networks may function in a completely different way. The behaviour of two neural network architectures will be described briefly in sections 2.1 and 2.2.

2.1 A single layer neural network

A neural network can be used to match patterns. A pattern may be an image, a fourier transform of a sonar signal, a radar echo, etc. The network can find the best matching pattern of a set of known patterns when presented with a pattern that it has never been presented with before (i.e. the input pattern). If the input pattern is a distorted version of one of the known patterns, the

network is able to recall the original pattern with which it corresponds. This type of neural network is known as a content addressable memory or Hopfield network, [Hopfield 1982].

The knowledge about the set of known patterns is represented in the network by the values of the weights of the connections between the PE's. The knowledge about each single pattern is distributed over all PE's in the network. The set of known patterns is learned by the network in a separate learning session.

Because all PE's in the network are fully connected to each other (figure 1a) there is no layered structure in the network, so the network is called a single layer network. A pattern is described using a fixed number of features. A feature may be a binary or an analog value. If a pattern is described using N features, the network contains N PE's. Each PE has one external input to read the value of the feature of the input pattern that corresponds with that particular PE. Each PE has N-1 internal inputs (one for each of the other PE's of the network) that are used read the outputs of these other PE's. All PE's execute the same simple processing equation. This equation is discussed in more detail in section 2.3 "Properties of a processing element". A Hopfield network with N PE's can contain 0.15 N known patterns without becoming unstable.

The network is operated as follows: first the input pattern is presented to the external inputs of the PE's of the network. Subsequently each PE calculates a new output value in parallel with the other PE's. Each PE may calculate its new output value asynchronously with respect to the other PE's. The calculation of a new output value uses the input values from the other PE's multiplied with the values of the corresponding weights and the value of a feature of the original unknown input pattern, also multiplied by a weight factor. The output values of the PE's keep changing, the weights remain fixed. After a number of iterations the network will have converged to a stable state. The outputs of the PE's then represent the best matching output pattern.

The weights are set (learned) in a separate learning phase. In this learning phase the set of patterns that must be stored in the network is repeatedly presented to the network, and the weights are adapted according to a learning rule (such as Hebbian learning or the delta rule). The learning phase will be discussed in more detail in section 2.4: "Learning a neural network".

2.2 A multi layer neural network

A neural network can be used to classify patterns. The network may for instance classify input patterns as belonging to class A or class B. The network consists of three layers: an input layer, a

hidden layer and an output layer (figure 1b). The network discussed here contains one hidden layer, more complicated networks may contain more hidden layers. It is a so called feed forward network. The PE's within a single layer are not connected to each other. The output of each PE of a layer is connected to an input of each PE of the next higher layer.

The input layer contains one PE for each feature that characterizes a pattern. The PE's in the input layer don't transform the input feature values. They just distribute the input feature values to the PE's of the first hidden layer.

A hidden layer is characterized by the fact that the PE's of a hidden layer are only connected to other PE's of the network and not to external inputs or outputs. The number of PE's in a hidden layer and the number of hidden layers is depends on the complexity of the classification problem. The PE's in the hidden layer(s) and in the output layer implement the same processing equation. In the output layer there is one PE for each class. See section 2.3 for more information about the dynamics of a PE. A network consisting of 50 PE's (34 PE's in the input layer, 14 PE's in the hidden layer and 2 PE's in the output layer) is already capable of simple SONAR signal classification [Gorman & Sejnowski 1988].

The network is operated as follows: a new input pattern is presented to the input layer. The feature values are distributed to and processed by the PE's of the hidden layer and finally distributed to and processed by the PE's of the output layer. The PE of the output layer with the highest output value represents the class to which the input pattern belongs.

The weights are set in a separate learning phase. In this phase examples of input patterns are presented to the PE's of the input layer while the class to which the input pattern belongs is indicated to the PE's of the output layer. This is called supervised learning. The weights can be set using a learning rule like Backpropagation, see [McClelland & Rumelhart 1986, pp. 318-364].

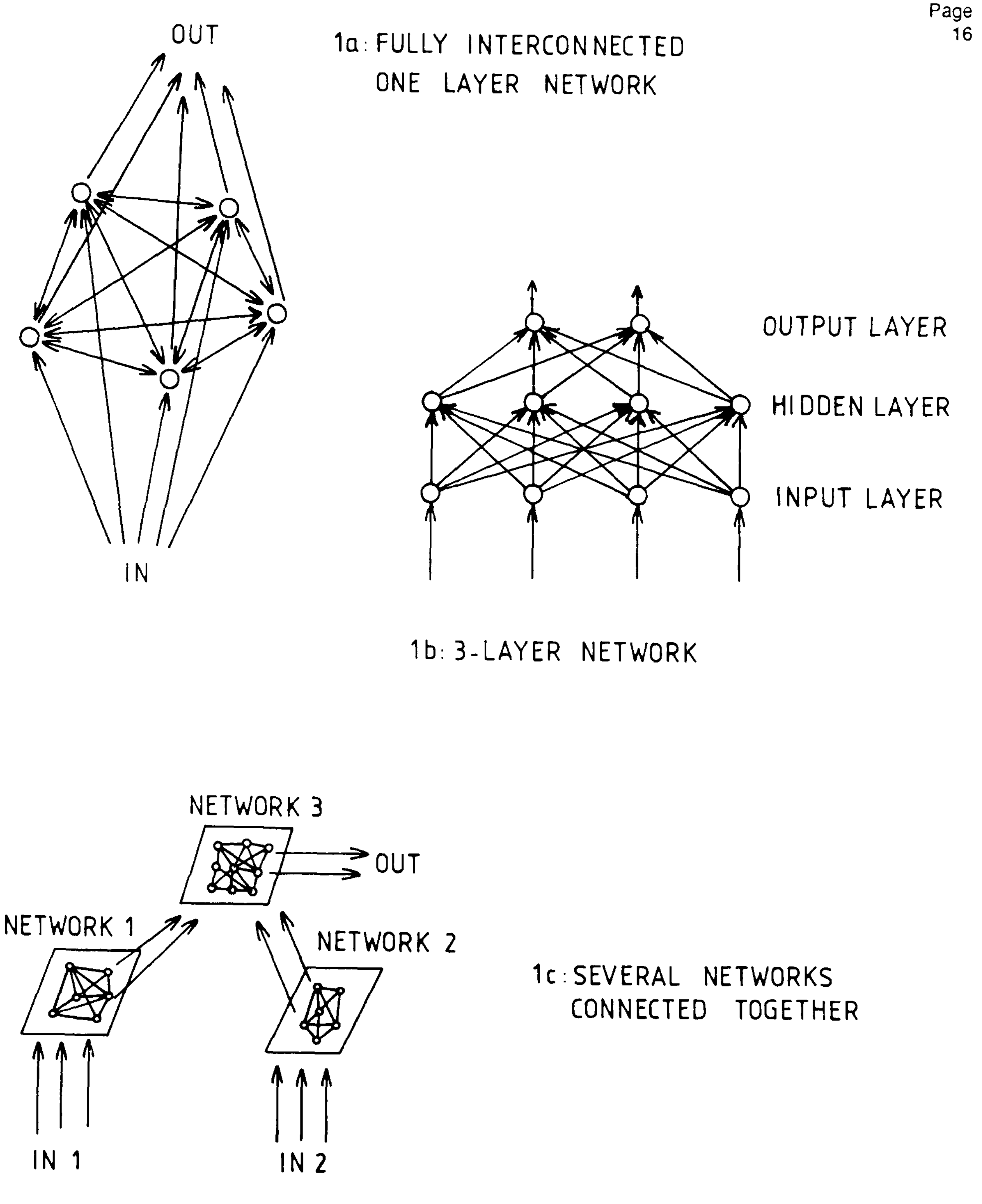

Figure 1 Conceivable neural network architectures

2.3 Properties of a processing element

A typical PE has many inputs (from other PE's or from the outside world) and a single output. There is a weight W_{ij} associated with the input from PE_i to PE_j. PE_i computes its output E_i as a simple (non-linear) function of the **weighted sum** of its N inputs (figure 2). Function Af(x) is also called the activation function of the PE.

$$E_i = A_f\left(\sum_{j=1}^{N} Q_j \cdot W_{ij}\right) \qquad (1)$$

$$A_f(x) = \frac{1}{1 + e^{-x}} \qquad (2)$$

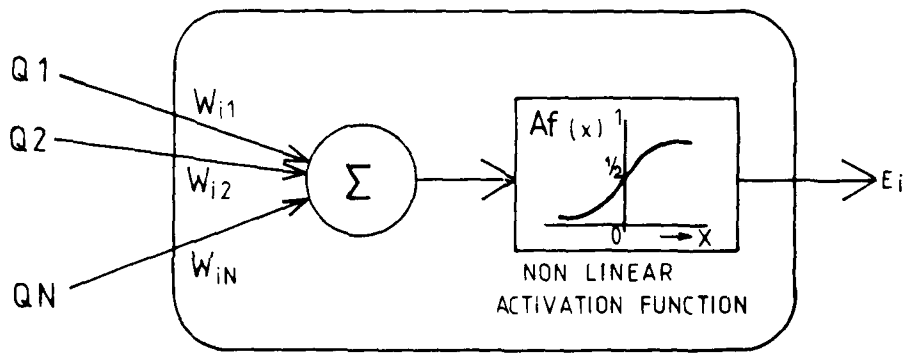

Figure 2 Schematic diagram of a processing element

A typical hardware implementation of a PE is non-linear, analog, has a low resolution and may be slow compared to digital integrated circuits. The state of the outputs of the PE's is regarded as the short term memory of the neural network. The short term memory changes very rapidly as the network iterates to a stable state.

The **knowledge** of a neural network is contained in the values of the weights of the PE's of the neural network. The state of the weights of the PE's is regarded as the long term memory of the neural network. The long term memory is adapted slowly using a learning rule. A positive

weight factor represents a positive or excitatory input. A negative weight factor represents a negative or inhibitory input. A zero weight factor is the same as "no connection".

A PE with N inputs divides its N-dimensional input space into two regions separated by an N-dimensional hyperplane that is realized by the activation function A_f of that PE. The output of the PE is active (high) in one of the regions and passive (low) in the other region. That is why such a region is called a **decision region**. For N=2 this is graphically shown in figures 3 and 4.

A single PE can successfully classify two classes that are linearly separable (figure 3). The output value OUT of a PE is the result of the activation function $A_f(x)$. This means that the range of OUT is limited to $0 < OUT < 1$. The PE is said to be passive (e.g. selecting class A) if $OUT < 0.5$ and active (e.g. selecting class B) if $OUT > 0.5$. The output of the PE is on the boundary of the decision regions A and B if $OUT = 0.5$. Now $A_f(x) = 0.5$ if $x = 0$. In the case of this simple 2-input PE the value of x is defined by equation (3).

$$x = Q_1 . W_1 + Q_2 . W_2 + \theta \qquad (3)$$

Here θ is a constant input to the PE. Input θ can be considered as a weight value W_3 connected to the constant value 1 ($\theta = W_3 . 1 = W_3$). This means that the value of θ can also be set in the learning phase. Now $x = 0$ if equation (3) equals zero. We can rewrite this as equation (4) to obtain the description of a line:

$$Q_1 = - \frac{W_2}{W_1} . Q_2 - \frac{1}{W_1} . \theta \qquad (4)$$

The output OUT of the PE will be active on one side of the line and passive on the other side. The position and orientation of the line in the 2-dimensional plane can be chosen arbitrary by choosing appropriate values for W_1, W_2 and θ. This line represents the hyperplane that separates the input into classes A and B in the simple 2-dimensional case. It can be seen that the X-or problem can't be solved using a single layer (= single PE) neural network because this problem is not linearly separable (figure 4).

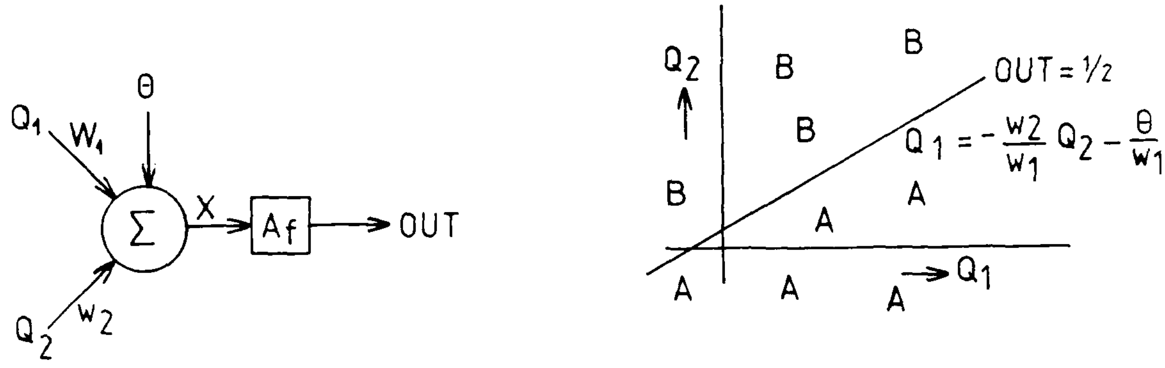

Figure 3 A line is a hyperplane in two dimensions

In the following discussion about the number of layers that is necessary to implement decision regions the input layer is not regarded as a processing layer. The two decision regions formed by a single neuron are both unbounded. A combination of N+1 hyperplanes can select a bounded decision region in an N-dimensional input space. This can be implemented by a network with a single hidden layer of N+1 PE's and an output layer consisting of a single PE that combines the hyperplanes formed by the hidden layer into one bounded region. Such a two layer network can also solve the X-or problem.

The use of more than N+1 hyperplanes for one decision region allows for a more flexible shape of that region. It is also possible to use a single PE in the definition of more than one decision region. In fact, this is what actually occurs in a real neural network.

Suppose that classes A and B are distributed over several separate regions. Each region is represented by a two layer network. We can then combine these regions by a PE of the layer on the next higher level. Such a three layer network can implement decision regions of unlimited complexity.

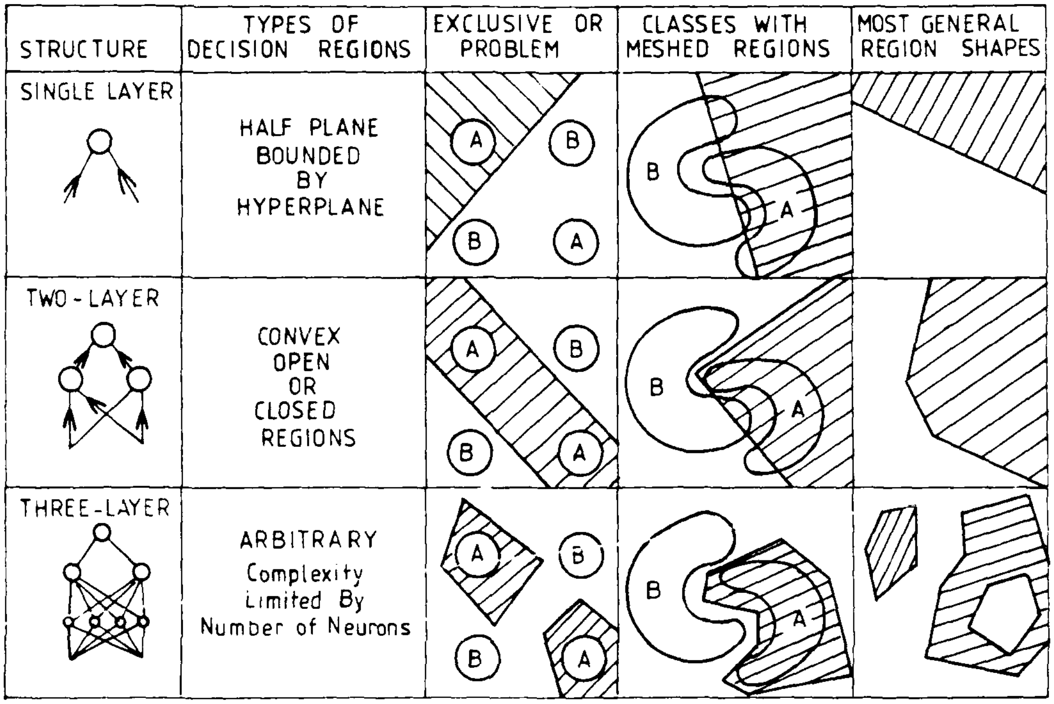

Figure 4 Decision regions in multi layer networks

2.4 Learning a neural network

The behaviour of a neural network is learned, it is not programmed. Learning means setting the weights of the PE's of the neural network in such a way that the neural network performs the desired function. The values of the weights can be calculated off-line (possibly on a separate computer system) or can be learned by the network itself using example patterns and a **learning rule**. The learning rule ensures that the neural network generates an internal representation of a solution to the problem to be solved.

A number of learning rules that are currently known and used:

- Hebb Rule [Hebb 1949], [Rochester, Holland, Haibt & Duda 1956] and [McClelland & Rumelhart 1986, pp. 35-76]
- Widrow-Hoff or Delta rule [McClelland & Rumelhart 1986, pp. 35-76]
- Generalized Delta Rule or Back Propagation [McClelland & Rumelhart 1986, pp. 318-362]
- Adaptive Resonance Theory [Grossberg 1980]
- Kohonen learning [Kohonen 1984]

With **supervised learning**, the neural network is presented with a series of inputs, while the corresponding **desired outputs** are presented at the teaching inputs. Supervised learning is used for associative memories or classifiers.

With **unsupervised learning**, the neural network is presented with a series of inputs only. There is no external "teaching input". Unsupervised learning is used to train nets to quantify vectors or to form clusters.

The learning data set (also called training data set) that is used to train a network is very important. All information that is necessary to solve a problem must be contained in the training set. If the set is presented to the network too often in the learning phase, the network may become 'overtrained' and lose its ability to generalize.

2.5 Application areas

Neural networks are especially useful in recognition and classification problems. These problems require fast evaluation of many hypotheses. Neural networks are capable of evaluating many hypotheses in **parallel**. These are just the problems that have not yet been solved satisfactorily using conventional computers but that can be solved readily using biological systems (like e.g. human beings). Neural networks are not very good at precise numerical calculations. A neural network would not be used to calculate the price of a car.

A short list of application areas:

- Scaling, translation and rotation invariant pattern recognition
- Production line control
- Inverse kinematics
- Optimization problems
- Distributed associative memory
- Multi sensor integration
- Hand written character recognition
- Radar pulse classification
- Text string recognition

2.6 Advantages of neural networks

The advantages of using neural networks extend beyond the promise of the huge computational power provided by massive parallelism.

Neural networks provide a large degree of ***fault tolerance***. Because a network consists of many PE's and even more connections and because the knowledge of the network may be distributed over all these PE's and connections, the failure of a few PE's or connections will generally not degrade the performance significantly. This is called **graceful degradation**. PE's that are already faulty before the learning phase begins are even less a problem: their outputs will just be ignored by the rest of the neural network.

The performance of a neural network is relatively independent of the accuracy of the PE's. The dynamic output range and internal calculations of a PE may be implemented using simple 8-bit arithmetic or crude analog circuits. The non-linear activation functions of the PE's of the network don't have to be exactly the same. The PE's of a neural network may be activated asynchronously with respect to each other. These facts allow the use of optical systems to implement massively parallel, robust and cheap neural networks. It is very difficult to harness the immense potentials of optical systems to implement conventional von-Neumann computers. The same facts allow the use of wafer scale technology for neural network implementation. Such a system will still work, regardless of the flaws that will probable be present on the wafer. The defective PE's will just be ignored.

Some neural networks have the ability to **continue learning** while performing their normal task. This is important because in many classification and recognition problems the environment of the network changes dynamically (while the network is in operation).

It is not always possible, or it can at least sometimes be very difficult and time-consuming, to create and verify an algorithm for a conventional von-Neumann computer for certain applications. Because a neural network is not programmed, this problem does not occur when using neural networks. In the learning phase the neural network builds an internal representation of the characteristics of its task. The learning data set must represent the environment in which the network will operate after the learning phase. Gathering this learning data set (together with choosing a neural network paradigm) is the task that replaces the classical algorithm design process.

2.7 *Disadvantages of neural networks*

The behaviour of a neural network is learned by presenting it with a number of examples. This behaviour is stored in the weights of the network using a learning process. If the learning data set is not carefully chosen then the required behaviour may not be learned.

It is sometimes necessary to find out why a system has produced a certain decision. The process leading to a decision produced by a program on a von-Neumann computer can always be traced, although this is sometimes not easy. The behaviour and knowledge of a neural network is stored in its weights. The PE's of a network may be updated in parallel and asynchronously. This means that it is often impossible to find out why a neural network reacts as it does. Each piece of knowledge of the network is distributed over all weights of the network. Although the neural network may perform one or more clearly defined functions, it may be impossible to locate a certain PE or weight in the network that performs a certain function. Neural networks can be used to solve problems for which it is very difficult or impossible to write an algorithm. It must not be expected that we can take a neural network, solve the problem and then use the information in the network to see **how** it solved the problem.

A neural network may not always provide the completely correct answer to a problem, although it will never generate a "divide by zero error" or something like that. Because the precise internal behaviour of a neural network is generally not known, it is also impossible to prove (mathematically or using software verification techniques) that a neural network system always produces a correct answer. Instead the performance of a neural network must be assessed

using statistical techniques. This may result in statements like "the network correctly classifies 96% of the patterns belonging to class A".

A neural network is not suited to perform precise numerical calculations. Most neural networks are not perfect. The decisions produced by a neural network may be near optimum instead of perfect. There is always a chance that a result produced by a neural network is wrong. This chance is dependent upon the kind of network, the number of PE's in the network, the set of examples, the learning rule, etc.

A neural network may not always detect the logic that is intrinsic in a problem. It may be possible to learn a network that 0+1=1, 1+1=2, 2+1=3, etc, but the network may still not be able to calculate 2+2=4. A neural network is not intelligent.

2.8 Creating a neural network

Instead of creating a program, a neural network engineer must accomplish the following steps to create a neural network:

- Define an appropriate neural network architecture
- Define a proper mapping from the environment to the inputs of the network
- Define a proper mapping from the outputs of the network to the environment
- Define the network parameters
- Choose the learning rule
- Gather a representative learning data set

2.9 Implementation of a neural network

Neural networks can be implemented in a number of ways. A short list of the possible implementations:

- Software simulation on an ordinary von-Neumann computer (ranging from a PC to a supercomputer).
- Software simulation on an adapted von-Neumann type computer (e.g with separate memory banks for fast parallel vector access, internal pipelines, special hardware, micro code optimized for neural network simulation) such as SAIC's Delta floating point processor and HNC's ANZA PLUS processor.

- Software simulation on existing parallel processing systems such as the NCUBE, the Connection Machine, transputer farms, etc.
- Hardware simulation using special purpose VLSI systems, such as Syntonic's Dendros chip-set or the neural bit slice from Micro Devices.
- Optical implementations using holograms. Current research investigates the use of alterable volume holograms to store the connection weights.
- Implementation using **bio-chips**. Biochips may be slower than current VLSI, but many more PE's could be implemented, resulting in a speed improvement of the system. Bio-chips may contain more faulty devices on a chip than current VLSI technology, but, as explained before, this does not have to be a problem for neural networks.

3 SPR NETWORK

This section first discusses the architecture of the SPR network. After that it is explained how this architecture is used to solve the name recognition problem. Some refinements and extensions to the SPR network concerning this particular application are discussed next. The implementation of the SPR network as a software simulation and on a parallel processing system is the last item of this section.

3 1 The architecture of the SPR network

A Spatio-temporal Pattern Recognition neural network is designed to recognize **time sequence relationships** in a series of input vectors $\underline{Q}$ (a vector $\underline{Q}$ consists of a number of analog or binary elements). The SPR neural network architecture is shown in figure 5. The SPR network is able to calculate a matching score between a series of input vectors and a number of series of vectors that are stored in the network. The network can be thought of as a bank of matched filters (each layer of the network represents one matched filter), where each filter is tuned to a specific series of input vectors.

The network consists of a number of layers. Each layer can recognize one series of input vectors. A layer contains a number of Processing Elements (PE's). A PE can calculate a **match value** between an input vector $\underline{Q}$ and a weight vector $\underline{W}$ stored in the PE. A PE has one input and one weight factor for each element of an input vector. The processing equations of a PE are given below. A graphical representation is shown in figure 6.

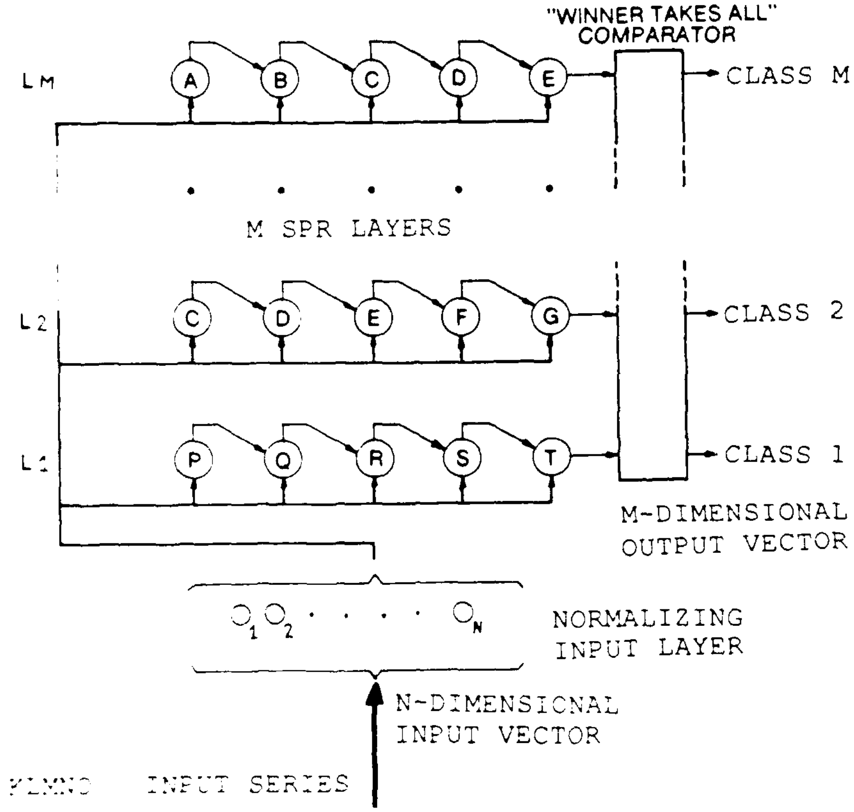

Figure 5 SPR network architecture

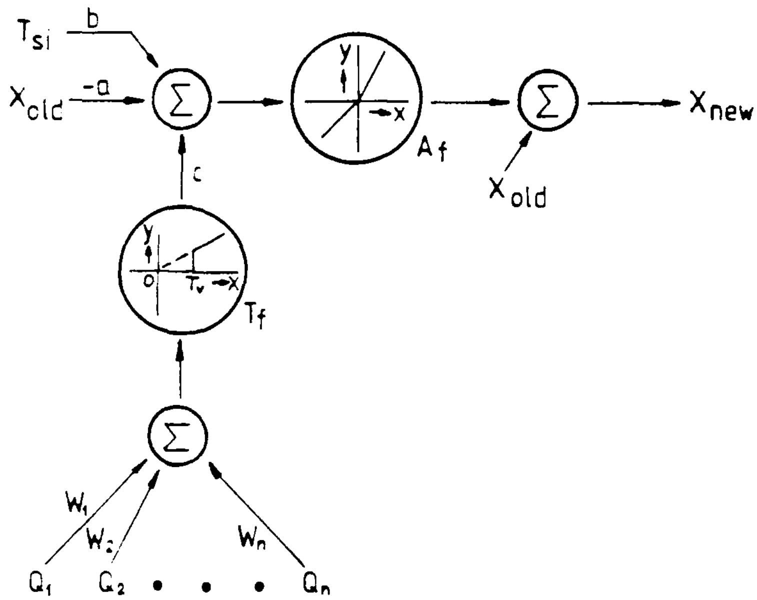

Figure 6 SPR processing equations

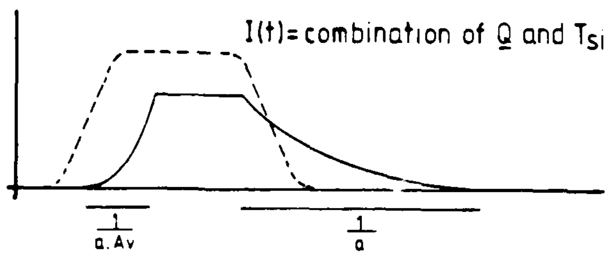

Figure 7 PE reaction to input

SPR Variables:

X_{new} = New output of PE (at time t+1)

X_{old} = Previous output of PE (at time t)

Tsi = Time sequence input (output from previous PE, at time t)

$\underline{Q}$ = Normalized input vector

$\underline{W}$ = Normalized weight vector

SPR Parameters:

Av	= Attack value	$Av \geq 1$
Tv	= Threshold value	$0 \leq Tv \leq 1$
a	= Decay constant	$0 < a \leq 1$
b	= Time sequence gain	$0 \leq b \leq 1$
c	= Input gain	$0 \leq c \leq 1$

SPR processing equations:

$$X_{new} = X_{old} + Af(-a.X_{old} + b.Tsi + c.Tf(\underline{Q}.\underline{W})) \tag{5}$$

$$Af(x) = \begin{cases} Av.x & \text{IF} \quad x > 0 \\ x & \text{IF} \quad x \leq 0 \end{cases} \tag{6}$$

$$Tf(x) = \begin{cases} x & \text{IF} \quad x \geq Tv \\ 0 & \text{IF} \quad x < Tv \end{cases} \tag{7}$$

Note that there is only **one** input from another PE (The Tsi input), and that that other PE is the previous PE in the **same** layer. Because of this the number of connections between PE's is O(N), where N = number of PE's.

If a PE is being activated by a constant input value I, which is a combination of Tsi and $\underline{Q}.\underline{W}$, it will reach its maximum output value with a time constant Tup (equation 8). When I = 0 the output will return to 0 with a time constant Tdown (equation 9). If Av > 1 then Tup < Tdown. This important task is performed by the **attack function** Af (see figure 7 and equation 6). The **threshold function** Tf (equation 7) keeps the PE from reacting if the match between $\underline{Q}$ and $\underline{W}$ is too small.

$$Tup = \frac{1}{a.Av} \tag{8}$$

$$Tdown = \frac{1}{a} \tag{9}$$

The input vectors are supplied to the SPR network **one after another**. **Each** input vector is presented to **all** PE's of **all** layers of the network **at the same time**. The value of the feed-forward connections between PE's in a certain layer indicates the match at that time between the series of input vectors that can be recognized by that layer, and the series of input vectors that is actually being offered to the network.

The important thing in the scheme described above is that the **sensitivity** of a PE to an input vector is controlled by the feed-forward output of the previous PE in the same layer.

As an example (*figure 8*) suppose that a layer is set to recognize a series of vectors (A, B, C), and that this series is supplied sequentially to the network at times T=1, T=2 and T=3. At T=1 PE A matches input vector A and is activated. At T=2 PE B matches input vector B and is activated. The output of PE B at T=2 is larger than the output of PE A at T=1, because it is activated by the input vector as well as the Tsi input from PE A. For the same reason, the output of PE C is even larger at T=3. So the outputs of the PE's in a layer keep getting larger as long as the input vectors keep matching. If there is a temporary mismatch between the series of input vectors and the PE's in a layer, e.g. if one of the input vectors is missing, this effect will persist, although the output value of the last PE of the layer will be less than if there would have been a perfect match. Note that the output value of a PE slowly decreases when its input is zero (look at PE A at T=1, T=2, T=3).

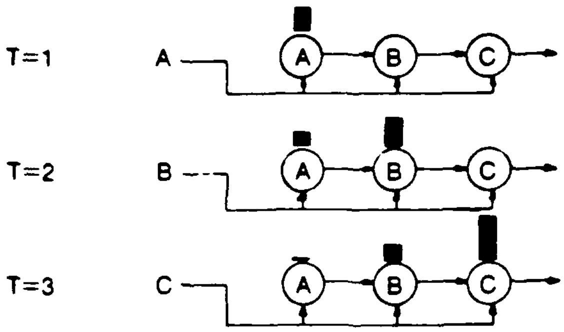

Figure 8 SPR matching

The output of the last PE of a layer is used as the **Match Value** (MV) between that layer and the input to the network. A comparator is used to choose the best-matching layer, or to provide a "no match" signal if all outputs of the layers stay below a pre-defined **Match Threshold** (MThrh) value.

The SPR processing equations resemble the nearest matched filter bank equations of [Hecht-Nielsen 1987] (figure 9). There are two important differences between these two neural network architectures with regard to the **connectivity**:

- The SPR network uses a fixed number (1 or 2) of feed-forward connections per PE. This fact reduces the number of connections from $O(N^2)$ for the matched filter bank to $O(N)$ for SPR. There are no connections between PE's in different layers. This fact is very important if the network is to be implemented using parallel processing hardware.
- The SPR network doesn't use the concept of a system activity threshold variable Γ. Γ occurs in the processing equations of each PE. Γ is defined as a function of the outputs of all PE's. This means that after each iteration Γ must be **recalculated** (requiring the availability of the output values of all PE's) and sent back to all PE's.

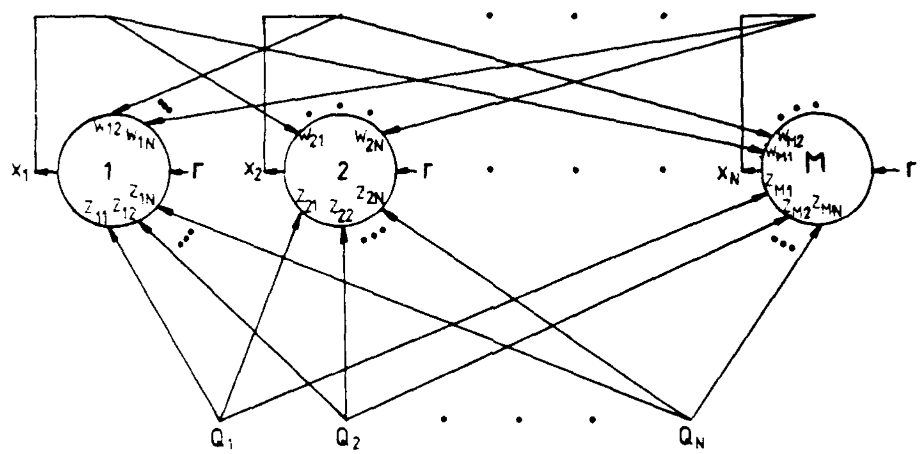

Figure 9 Nearest matched filter bank

Nearest matched filter bank variables:

$X_i(t+1)$	= New output of PE	
$X_i(t)$	= Previous output of PE	$0 \leq X_i(t) \leq 1$
$I_i^1(t)$	= Time sequence activation	
$\underline{W}_i$	= Time sequence weight vector	$0 \leq W_{ij} \leq 1$, $W_{ii} = 0$
$I_i^2(t)$	= Input vector activation	
$\underline{Q}(t)$	= Input vector	$\lvert\underline{Q}\rvert = 1$
$\underline{Z}_i$	= Pattern sample weight vector	$\lvert\underline{Z}_i\rvert = 1$
$\Gamma(t)$	= System activity threshold	$\Gamma(t) \geq 0$
$S(t)$	= System power level	

Nearest matched filter bank parameters:

a	= Decay constant	$a > 0$
b	= Overall PE input	$b > 0$
c	= Attack constant	$0 \leq c \leq 1$
α	= α-β controller parameter	$\alpha \geq 1$
β	= α-β controller parameter	$\beta \geq 1$
T	= Power level target	
m	= Number of PE's	$m \leq n.N$
n	= Number of sample points per pattern	
N	= No. of stored spatio-temporal patterns	

Nearest matched filter bank processing equations:

$$X_i(t+1) = X_i(t) + A(-a.X_i(t) + b(I_i(t)^1 + I_i^2(t) - \Gamma(t)^+) \tag{10}$$

$$I_i(t)^1 = \sum_{j=1}^{m} W_{ij}.X_j(t) = \underline{W}_i.\underline{X}(t) \tag{11}$$

$$I_2^1(t) = \sum_{j=1}^{n} Z_{ij}.Q_j(t) = \underline{Z}_i.\underline{Q}(t) \tag{12}$$

$$\Gamma(t+1) = \Gamma(t) + \alpha.(S(t) - T) + \beta.S'(t) \tag{13}$$

$$S(t) = \sum_{i=1}^{m} X_i(t) \tag{14}$$

$$(u)^+ = \begin{cases} u & \text{IF } u > 0 \\ 0 & \text{IF } u \leq 0 \end{cases} \tag{15}$$

$$A(u)^+ = \begin{cases} u & \text{IF } u > 0 \\ c.u & \text{IF } u \leq 0 \end{cases} \tag{16}$$

3.2 Name recognition with a SPR neural network

A text string is considered as a sequence of characters. Each character of a text string is represented by one vector. A string consisting of N characters is represented by a series of N vectors. SPR considers a string as a time sequence because the vectors are supplied to the network one after another in the order in which they appear in the text (from left to right).

Each PE contains a **weight vector** $\underline{W}$ that represents the character(s) that is/are to be recognized by that PE. If a PE is used to recognize only one character, its weight vector is identical to the vector representation of that character.

A **pre processor** fetches each message and supplies the words of that message to the network one by one.

Data representation is an important subject in neural network theory. SPR networks need **normalized** input and weight vectors. This is necessary because the match between the input vector $\underline{Q}$ and the weight vector $\underline{W}$ is calculated using the dot product (equation 17).

$$\underline{X} \cdot \underline{Y} = |\underline{X}| \cdot |\underline{Y}| \cdot \cos(\theta) \qquad (17)$$

This match must correspond with the value of $\cos(\theta)$, so $|\underline{X}|$ and $|\underline{Y}|$ must be 1, i.e. $\underline{X}$ and $\underline{Y}$ must be normalized.

The characters can be represented by an 8-bit vector (e.g. using the 8-bit ASCII code). This vector is normalized by a normalizing input layer and then supplied to the network.

3.3 Refinements and extensions

The bare-bones SPR network architecture as discussed above is suitable for the name recognition problem, but **not ideal**. Some refinements and extensions can be made to the network to let it perform even better. The idea is to put as much **a priori** knowledge about the name recognition problem into the architecture of the network as possible.

3.4 An extra Tsi input

To make the network less sensitive to a pair of interchanged vectors in a series of input vectors, an extra Tsi input is added to each PE (figure 10). Suppose a layer contains P PE's. Then PE[N] has a Tsi_1 from PE[N-1] and a Tsi_2 from PE[N-2]. The PE activation function now looks like:

$$X_{new} = X_{old} + Af(-a.X_{old} + b_1.Tsi_1 + b_2.Tsi_2 + c.Tf(\underline{Q}.\underline{W})) \quad (18)$$

Suppose that a layer is set to recognize the name 'ABCDEF', and that the input name is '**ACB**DEF'. PE D will be activated by an input 'C' (via Tsi_1, as usual) as well as by an input 'B' (via Tsi_2). This way, the **order** of the two inputs 'B' and 'C' is less important for the activation of PE D.

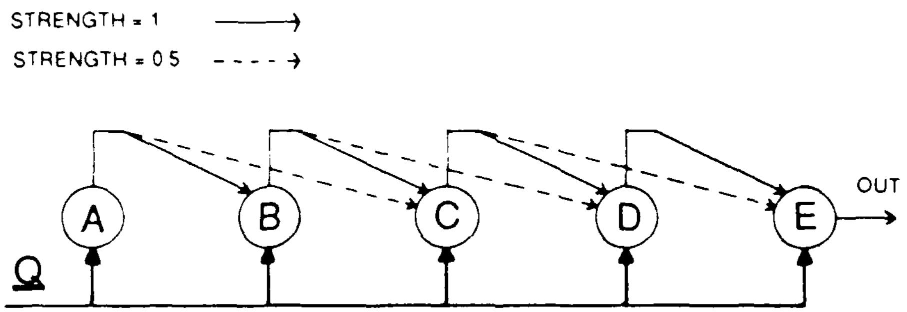

Figure 10 An extra TSI input

3.5 Normalization of the match value of a layer

In a standard SPR network each layer has the same amount of PE's. In this application, the names to be recognized may have different lengths. If a layer has P PE's and the name to be recognized by the layer is C characters long, than the weights of the first (P-C) PE's in the layer are set to zero.

Suppose we have two layers, one to recognize a name of length 3 (called L3) and one to recognize a name of length 6 (called L6). In the case of a perfect input for L3 as well as L6 the match value (MV) of L6 will be larger than the MV of L3, because the outputs of the PE's in L6 have had more network iterations (6 instead of 3) to grow. So there is a difference in MV, although the inputs are both perfect.

This problem is solved by **normalizing** the MV's of all layers. The maximum MV for a layer of length N, called MMV_n, is known (it can be calculated off-line or we can just input a perfectly matching string into the network and read the value of PE N after N iterations). An extra **Output PE** (OPE) is added to each layer (figure 11). The OPE multiplies the output of the last PE in a layer of length N with a factor $1/MMV_n$. This way, the output of the OPE is always 1 for a perfect input, **independent** of the length of the name.

max. output of PE 'E' = MMV_5

OPE : Output = Input x $1/MMV_5$

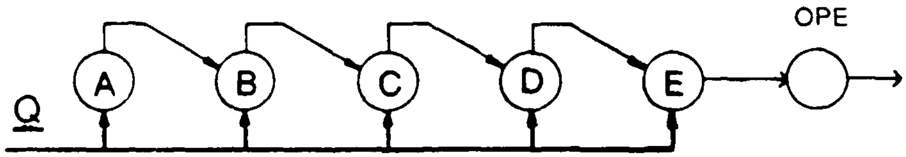

Figure 11 Adding an output OPE

3.6 Remembering the maximum PE output value

Until now it was assumed that the output value of the **last** PE in a layer gives the best indication for the match between the input name and the name stored in that layer. If one **fixed** PE must be chosen, out of all available PE's in a layer, to indicate the match, the last PE is indeed the best choice. However, this strategy is not perfect. A better strategy is discussed below (figure 12).

Suppose a layer L is set to recognize the series S='HARLINGEN'. The '#' character is used to indicate a character that does not match with any PE in any layer. There are two inputs: A='HARLI####' and B='#####NGEN'. Input A **should** match better than input B because A has 5 characters in common with S and B has only 4 characters in common with S. With the network as described until now, this will not happen. If we look at the output of the last PE of L, B will score better than A.

Suppose A is presented to the network. The first 5 PE's of the layer will show an increasing output value, because the first 5 characters of A match with S. The last 4 PE's will show a decreasing

output value, because the last 4 characters of A don't match with S. Let FO_A be the Final Output reached (i.e. that of PE 9). Let MO_A be the Maximum Output value reached (i.e. that of PE 5). Then $FO_A < MO_A$. In figure 12 $FO_A = 1.0$ and $MO_A = 2.3$.

The following will happen when B is presented to the network: The outputs of the first 5 PE's of the layer will stay zero because the first 5 characters of B don't match with S. The last 4 PE's will show an increasing output value, because the last 4 characters of B match wit S. Let FO_B be the Final Output reached (i.e. that of PE 9). In this case, $MO_B = FO_B$. In figure 12 $FO_B = MO_B = 1.9$.

The result is that $FO_B > FO_A$ (although $FO_B < MO_A$), indicating B as matching better than A. To handle this problem correctly, a Maximum Processing Element (MPE) is added to each layer. The MPE is connected to the outputs of all PE's in that layer and **retains** the maximum output that ever occurred in that layer (figure 13). The output of the MPE is then used instead of the output of the last PE of the layer (figure 12). Using the MPE, the problem is solved because now $MO_A > MO_B$ and input A wins.

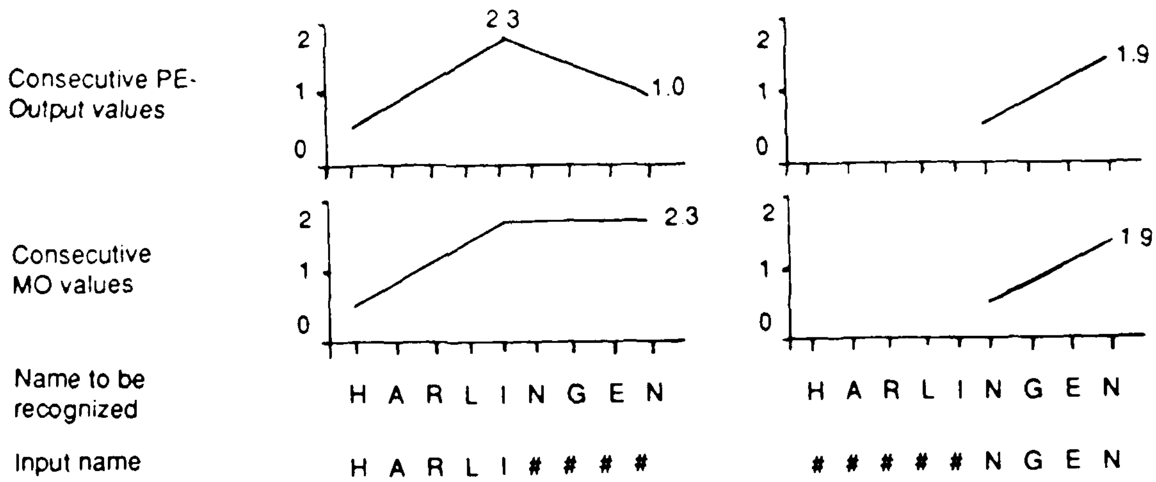

Figure 12 Effect of the MPE

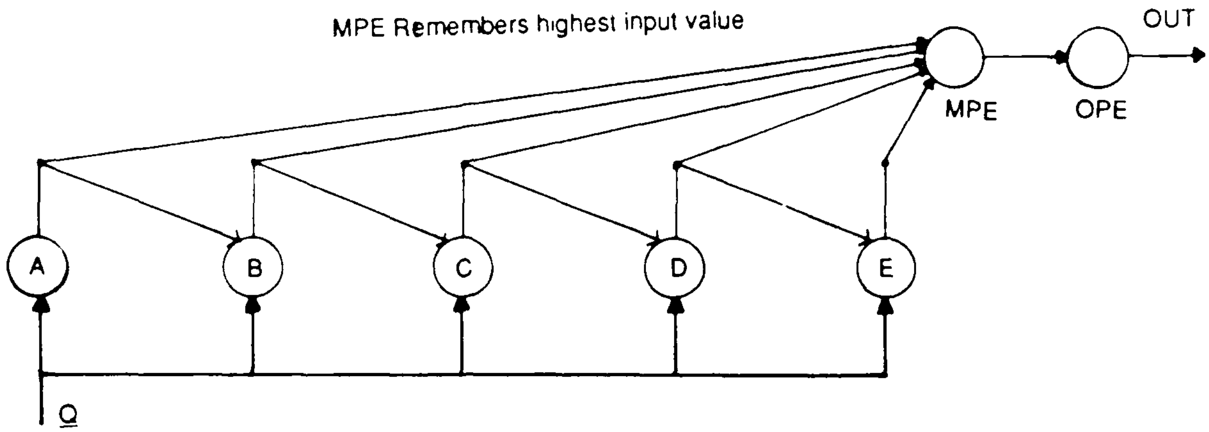

Figure 13 Adding a Maximum-PE (MPE)

3.7 Implementation as a software simulation

A program called 'SPR' has been written. The program simulates a SPR network used to recognize names as described in this report, including the refinements and extensions discussed above. The program is coded in Turbo Pascal 5.0 and runs on a PC AT or compatible, optionally using a 80287 coprocessor. The program consists of 4000 lines of source code.

The SPR program provides an **interactive** SPR network test and development environment. It is possible to run the network, change the parameters, run the network again, store the results, recall, learn, and store the weight vectors, etc. The program can generate a complete overview of what happens inside a network while it is running, numerically as well as graphically.

Each application of an SPR network requires its own **optimum** set of network parameters (Av, Tv, a, b1, b2, c). It is a known problem that an optimum set for these kind of networks is not easily found. The SPR program offers the possibility to adapt the SPR parameters to try to reach an optimum set. This is done using a technique called **simulated annealing** [Aarts & Korst 1987].

Suppose we do a total of N parameter updates, with $N = O(10^3)$. For each update, one of the 6 parameters of the parameter set is chosen randomly (so each parameter is updated N/6 times on the average). This parameter is updated by adding a signed random number R to it. The performance of the network is assessed (using a random character distortion measurement as

discussed in the chapter 'Results'. If the performance of the network is better with the updated parameter than it was before, the value of the updated parameter will be accepted as the new parameter value with a chance P (with P somewhere in the order of 90%). Because P < 100% a parameter value may be accepted with a chance (100 - P)% even if it makes the performance worse. This is to make it possible to escape out of local minima. The absolute value of R decreases gradually during the series of parameter adaptations. This is equivalent to lowering the 'computational temperature' of the random update process.

The above strategy assumes that a single parameter set (Av, Tv, a, b1, b2, c) is used for all layers in the network. Another approach would be to tune the parameter set (Av^k, Tv^k, a^k, $b1^k$, $b2^k$, c^k) for each layer K independently. This strategy would allow each layer K to recognize its input maximally. However, an optimum recognition for a single layer of a single name that allows as much distortion as possible, is to just always recognize the input as correct, because each conceivable input can always be formed by applying a number of distortions to the original name. The parameter set for each layer depends on the names that must be recognized by that layer **and** by the names that must be recognized by the other layers of the network. So it is not possible to set the parameter set for one layer independently of the other layers. This does not mean that the parameter set should be the same for each layer. It might be possible to devise a parameter setting algorithm that produces an optimum non identical parameter set for each layer of the network. However, this subject has not been investigated further.

When simulating a network of 11 names, each consisting of 12 characters, the program can process two names per second. This number can be increased by removing the book-keeping tasks that are necessary to look at what happens 'inside the network". This was not done, because that is just what the simulator is used for. Assembly routines could be used at critical points.

The speed of a neural network can be defined as the number of connections between PE's that can be processed in one second. A good way to increase the performance (speed) of the program is to interface it to the ANZA Plus neural network simulator (from HNC incorporated), which can process $O(10^6)$ network connections per second, as opposed to $O(10^4)$ connections per second using an IBM AT. A still better way is of course to really implement the network on a parallel processing system, as discussed below.

3.8 Implementation on a parallel processing system

A SPR network is very well suitable for implementation on a parallel processing system. Because there is no inter-layer communication, each processor of the system can implement one or more layers of the SPR network without any communication between the processors. A parallel implementation of the SPR network does not suffer from the **Interconnection bottle-neck**, because the number of connections in the network is of O(N) and not of O(N^2), where N is the number of PE's. This means that the SPR architecture is **scalable**. The speed of the system increases linearly with the number of processors in the system. We are thinking of implementing an SPR network using transputers [INMOS 1988] (figure 14).

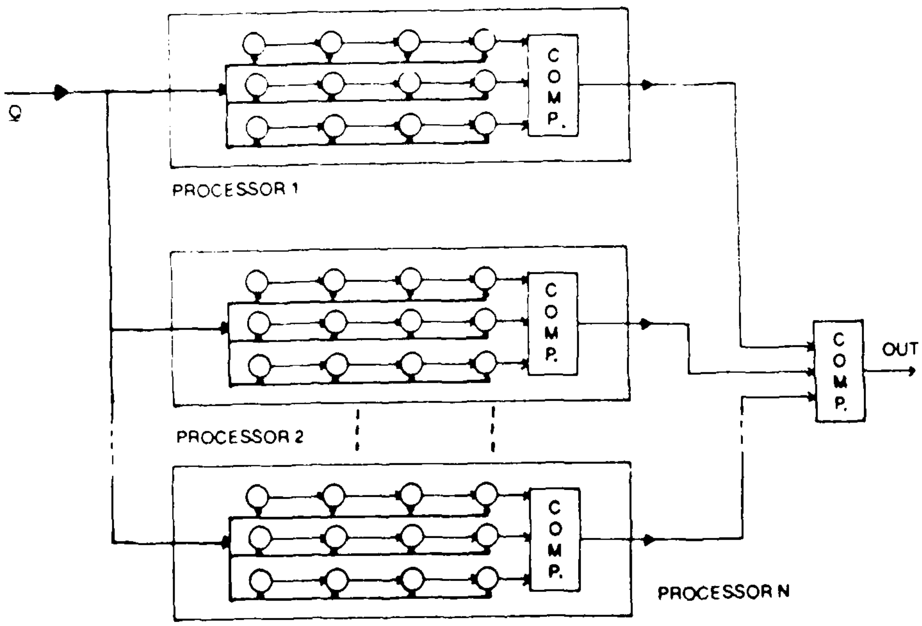

Figure 14 Parallel implementation of SPR network

4 WEIGHT VECTORS

Once the architecture of the network and the processing equations of the PE's have been determined, the weights must be set. The weights must be set according to the purpose of the network. Sometimes the weights can be calculated off-line. Most neural network architectures are related with some kind of learning rule. This learning rule is used to set the weights in one or more learning sessions. Learning can be done supervised or unsupervised. The weights for the name recognition SPR network can be calculated **off-line** or they can be set using **supervised** learning.

4.1 Off-line calculation

Off-line calculation of the weights is very simple: the weight vector of a PE is assumed to be the same as the vector representation of the character that must be recognized by that PE.

Suppose that the name 'ALKMAAR' is the N-th name that must be recognized by the network. The weights of the PE's of the N-th layer are set to recognize this name. Suppose that each layer contains P PE's. The name 'ALKMAAR' has 7 characters, so the weights of the last 7 PE's of layer N must be set. The weights of PE's [N,1] to [N,P-7] are set to zero. The weights of PE [N, P-6] are set to the vector representation of 'A', those of PE[N, P-5] to 'L', etc. The weights of PE [N,P] are set to 'R'.

4.2 The learning rule

A supervised learning rule is used. The supervision consist of the fact that only those words of a message that represent names of ships are selected and used to train the network.

The learning rule assumes a network of M layers. Initially it is assumed that the number of names to be recognized is R, where $R \leq M$. The learning rule allocates one layer for each name to be recognized. When a name with a specific error is encountered often, an extra layer may be allocated for that name. This layer will then be set to match that name while it contains the error. When this happens there will be L layers ($L \geq 2$) that refer to the same name. The outputs of these L layers must then be merged together by a **post processor**, so that the correct name will be indicated when one of these L layers wins the competition. This information must be provided by the learning supervisor.

When there are more names to be recognized by the network than there are layers available (i.e. when R > M), the M names that occur most frequently will be stored. This procedure is discussed in more detail below (figure 15).

If MV > MThrh (MV = match value, MThrh = match threshold), it is assumed that the input name is similar (but not necessarily identical) to the name represented by the winning layer in the network, and the weights $\underline{W}$ of that layer are **adapted** according to the vectors $\underline{Q}$ of the input name. For each character vector $\underline{W}$ of the winning layer this adaptation consists of adding a fraction f of the input vector $\underline{Q}$ to $\underline{W}$ and normalizing the result (equation 19).

$$\underline{W}new = |(\underline{W}old + f\underline{Q})| \tag{19}$$

If the input name is identical to the name to be recognized by the winning layer, effectively no adaptation will be performed (because for each character of the input name $\underline{Q}i = \underline{W}i$).

If MV < MThrh, it is assumed that the input name is not yet stored, and a **new layer** is allocated to refer to this name.

If a new layer must be allocated, but all layers of the network are already in use, the layer that least frequently won the competition in the past is **cleared** and used for the new name. This means that the **activation history** of each layer must be kept track off. The layer with the lowest activation history is cleared.

This strategy prevents the network from the **"information overflow"** problem, a problem that occurs e.g. in a Hopfield network when one tries to store more than about 0.15 N (N = the number of PE's) classes in the network. The performance of a Hopfield network will **degrade** rapidly if this occurs. The SPR learning strategy resembles Grossberg's ART algorithm [Grossberg 1980] in this respect.

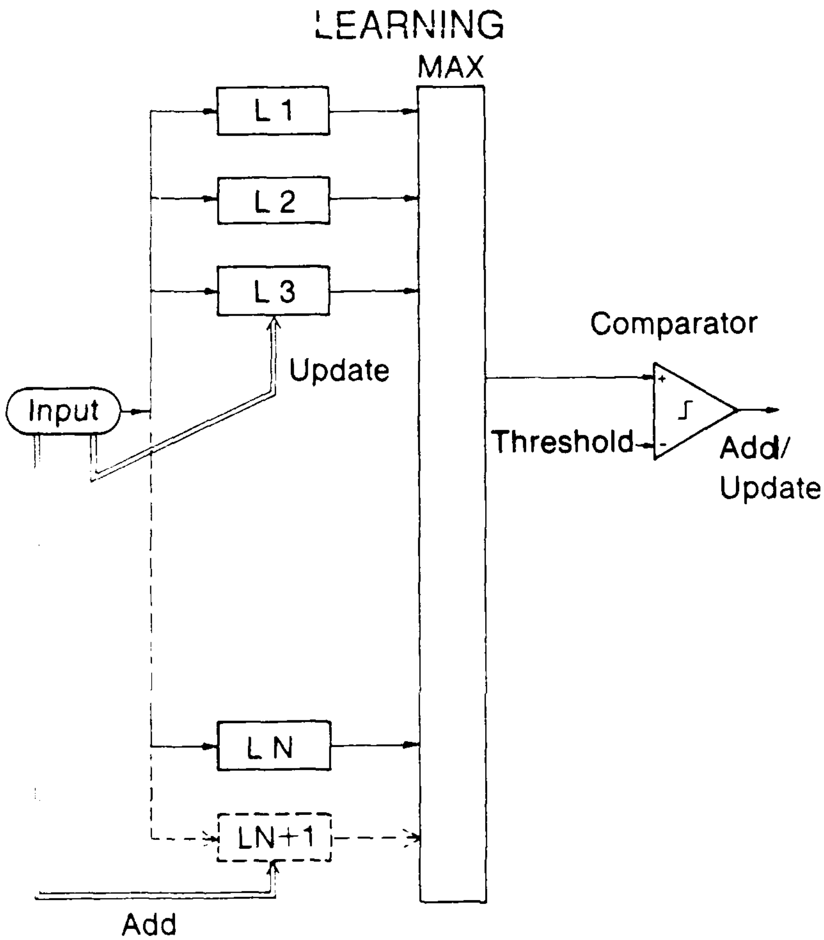

Figure 15 Allocation of layers in a SPR network

5 RESULTS

The results of two measurements of the performance of the SPR network are discussed below. The test network is set to recognize 11 names. A name can consist of up to 12 characters.

The network parameters are set to the following values:

Av	= Attack value	= 1.3
Tv	= Threshold value	= 0.5
a	= Decay constant	= 0.8
b_1	= Tsi 1	= 0.4
b_2	= Tsi 2	= 0.3
c	= Input gain	= 1.0

The names used to test the network with are:

ALKMAAR
DELFZYL
DORDRECHT
HARLINGEN
MAASSLUIS
SCHEVENINGEN
SCHIEDAM
URK
VLAARDINGEN
WILLEMSTAD
ZIERIKZEE

The score of each of the 21 measurement points of a graph is calculated by distorting and processing each of the 11 names 5 times. A total of 21 x 11 x 5 = 1155 runs of the network must be performed for a single graph.

One run of the network consists of processing one name. Processing one name means processing each of its L characters one at a time. Each character is processed by all PE's of the network. Processing a character means calculating the dot product of two vectors of N elements, i.e. performing N multiplications. If the network can recognize M names, the number of PE's NPE

in the network is NPE = M x L. The Number of Multiplications NM needed to process one name is then:

$$NM = L \times NPE \times N = L \times M \times L \times N = L^2 \times M \times N \qquad (20)$$

With L = 12, M = 11 and N = 8 (ASCII code), NM = 12,672. The Total Number of Multiplications TNM for one graph is then: TNM = 1155 x NM = 14,636,160.

Two different types of distortions were used, resulting in two different measurements, both of which are discussed below.

5.1 Random character distortion

The first measurement uses a random character distortion of some of the characters of the input names. Each of the characters of each input name has a chance P (in %) to be distorted. The distortion consists of choosing a random character instead of the original character. P varies from 0% to 100% in increments of 5%. The score for a percentage P is the percentage of the 55 names that is recognized correctly.

Examples:

```
'ABCDEFGHIJ' & P = 20% -> 'ABRDEFGAIJ'
'ABCDEFGHIJ' & P = 50% -> 'AXJDPFGAIM'
```

The test shows that the network still recognizes 95% of the names when 30% of the characters of an input name is distorted. The performance decreases gradually when P increases (figure 16).

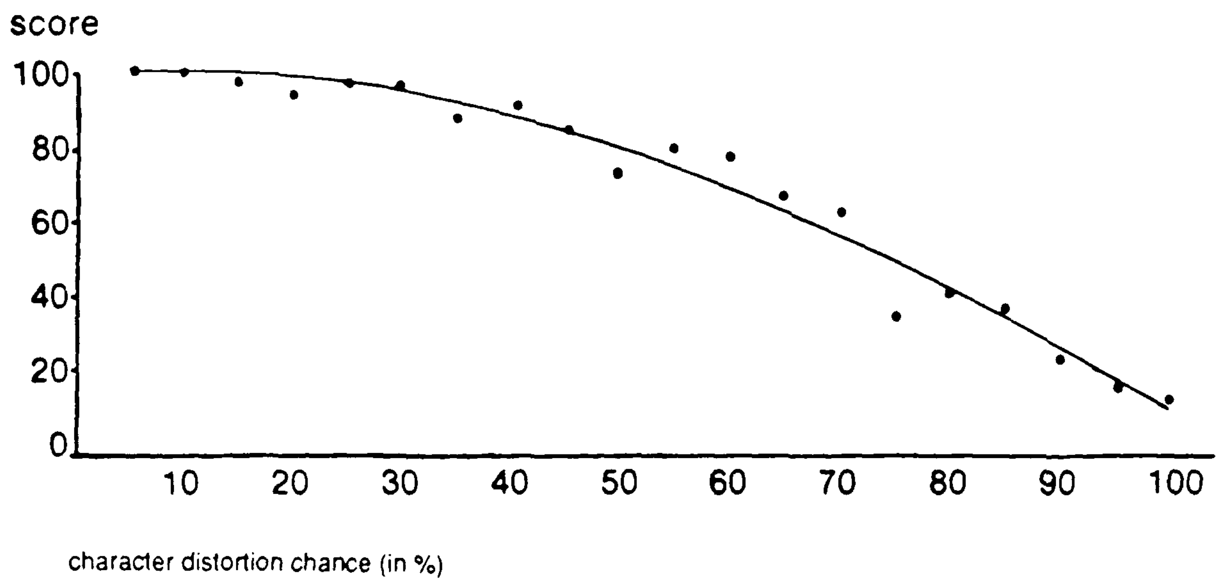

Figure 16 Random character distortion

5.2 Random character interchange

The second measurement uses a character interchange of some of the character pairs of the input names. The string 'ABCD' consists of 3 character pairs: 'AB', 'BC' and 'CD'. Each of the character pairs of each input name has a chance P (in %) to be distorted. The distortion consists of **interchanging** the two characters of a character pair. P varies from 0% to 100% in increments of 5%.

Examples:

```
'ABCDEFGHIJK' & P = 20% -> 'ACBDEFGIHJK'
'ABCDEFGHIJK' & P = 30% -> 'BACEDFGHJIK'
```

If the score is calculated as above, i.e. as the percentage of the 55 names that is recognized correctly, the score is always greater than 98%, even with 100% of the character pairs interchanged. This is partly because the network is indeed very insensitive to character pair

interchange and partly because the test network can choose only from 11 names (the chance of choosing the wrong name will increase if there are more names to choose from).

The random character interchange measurement requires a more discriminating method to calculate the score. Let AVB = Activation Value of the Best matching layer and AVS = Activation Value of the Second best matching layer. The score for one input name is calculated as the **relative difference** between the activation values of the best matching layer and the second best matching layer (equation 21).

$$\text{score} = \frac{\text{AVB-AVS}}{\text{AVB}} \qquad (21)$$

The resulting score of a measurement consisting of processing the 55 names with a character pair interchange chance P is the mean value of the scores for each of the 55 names (figure 17).

The test shows that the network discriminates quite well between the best and second best matching layers (the score stays between 0.65 and 0.45) and decreases gradually when P increases.

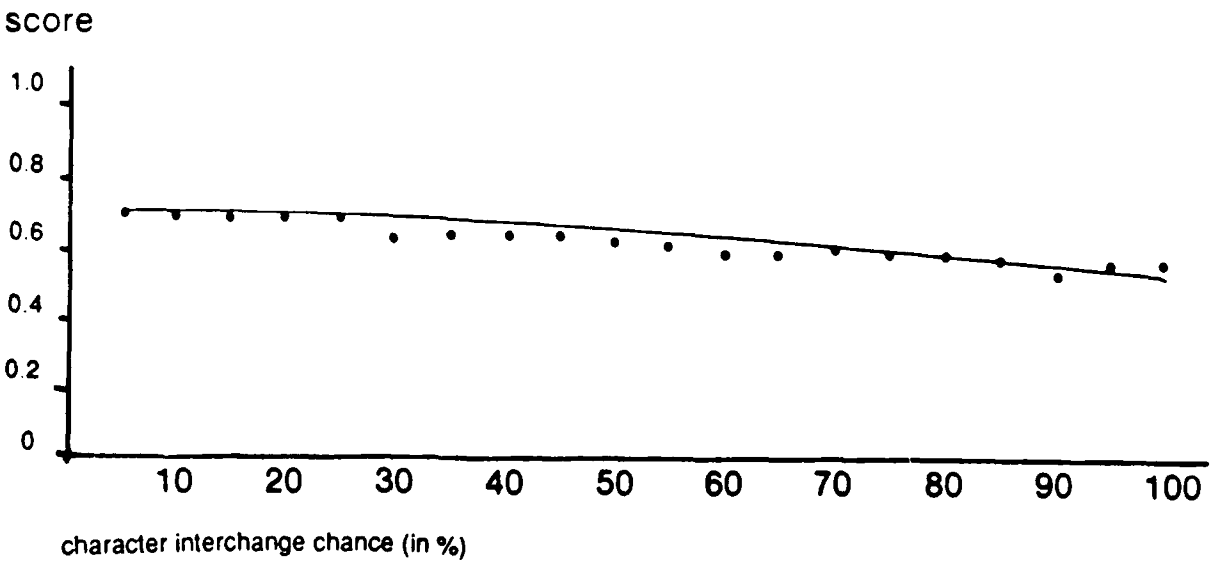

Figure 17 Random character interchange

6 CONCLUSIONS

- The SPR network works well. It recognizes names when one or more of the characters are erroneous, missing, interchanged, or when extra characters are added.

- The use of a SPR network for name recognition gives good results while the paradigm itself is **simple**. "Information overflow" can not occur in the learning phase. If the capacity of the network is too small, only the most frequently used names are learned.

- A working implementation of the SPR network for the name recognition problem has been realized. The program runs on an IBM PC or compatible.

- Because the SPR architecture is inherently parallel the SPR network is suitable for implementation on a parallel processing system. Because there is no inter processor communication while processing the layers of the network, the speed of the system increases **linearly** with the number of processors, i.e. the SPR architecture is linearly scalable.

- Comparable results can be obtained using expert systems. Expert systems require the existence of a rule base, which must be created by the designers of the system. Creating a rule base can be very difficult. All possible situations must be analyzed. A neural network generates its own internal representation of the characteristics of its input. This approach only works if the learning data set is a close approximation of the real world environment, in which the network is supposed to function.

P.L.J. van Lieshout
(Groupleader)

P.P. Meiler
(Author)

7 REFERENCES

Aarts E.H.L. & Korst J.H.M., *Combinatorial Optimization on a Boltzmann Machine*, Philips Research Laboratories, Eindhoven, The Netherlands. Submitted to Journal of Parallel and Distributed Computing, 1987.

Duda W.L., Haibt L.H., Holland J.H., Rochester N., *Tests on a cell assembly theory of the action of the brain, using a large digital computer*, IRE Transactions on Information Theory IT-2, pp. 80-93, 1956

Gorman R.P., Sejnowski T.J., *Analysis of Hidden Units in a Layered Network Trained to Classify Sonar targets*, Neural Networks, Volume 1, Number 1, Pergamon Press, pp. 75-89, 1988

Grossberg S., *How does a brain build a cognitive code?*, Psychological Review 87, pp. 1-51, 1980

Grossberg S., *Nonlinear Neural Networks: Principles, Mechanisms, and Architectures*, Neural Networks, Volume 1, Number 1, Pergamon Press, pp. 17-61, 1988

Hebb D.O., *The organization of behaviour*, Wiley, New York, Introduction and Chapter 4, *The first stage of perception: growth of the assembly*, pp. xi-xix, 60-78, 1949

Hecht-Nielsen R., *Nearest matched filter classification of spatiotemporal patterns*, Applied Optics, Vol. 26, No 10, pp. 1892-1899, May 15, 1987

Hopfield J.J., *Neural networks and physical systems with emergent collective computational abilities*, Proceedings of the National Academy of Sciences 79, pp. 2554-2558, 1982

INMOS, *The transputer family*, INMOS limited, Bristol, UK, 1988

Kohonen T., *Self-organization and Associative memory*, Springer-Verlag, Berlin, 1984

Lippmann R.P., *An Introduction to Computing with Neural Nets*, IEEE ASSP Magazine, pp. 4-22, April 1987

McClelland J.L. & Rumelhart D.E., *Parallel distributed processing*, Vol. I, II and III, Cambridge MA., MIT press, 1986

Wezenbeek van A., *BSB radar pulse classification*, report number FEL-90-B023, TNO-FEL, 1990.

8 LITERATURE

Bakkers A.P., *Neural controllers and transputers*, Control laboratory, Electrical engineering department, University of Twenthe, The Netherlands. Presented at the BIRA seminar, Antwerp, Belgium, October 1988.

Bengio Y., Cardin R., Mori de R. & Merlo E., *Programmable execution of multi layered networks for automatic speech recognition*, Communications of the ACM, Vol. 32, Number 2, pp. 195-199, February 1989.

Berkel van B. & Smedt de K., *Triphone analysis: a combined method for the correction of orthographical and typographical errors*, Institute for Applied Computer Science, TNO-ITI, Delft, The Netherlands, 1987.

Bharath R., *Information Theory*, Byte, pp. 291-298, December 1987

Chalmers E.G. & Paddon D.J., *A system configuration for very large database problems*, Department of Computer Science, University of Bristol, Bristol BS8 1TR, England. Published in the proceedings of the 11th Occam User Group technical meeting, Edinburgh, Scotland, September 1989

Coolen A. & Kuijk F.W., *A learning mechanism for invariant pattern recognition in neural networks*, Dept. of Medical and Physiological Physics, University of Utrecht, The Netherlands. Presented at the Neural Networks Workshop, Utrecht, The Netherlands, February 1989.

Grossberg S., *Adaptive pattern classification and universal recoding: I. Parallel development and coding of neural feature detectors*, Biological Cybernetics 23, pp. 121-134, 1976

Hanazawa T., Hinton G., Lang K., Shikano K. & Waibel A., *Phoneme recognition using time-delay neural networks*, IEEE transactions on Acoustics, Speech and Signal Processing, Vol. 37, No. 3, pp. 328-339, March 1989.

HNC incorporated, *ANZA Plus User's Guide and Neurosoftware documents*, Release 2.1, San Diego, California, 1988

Houtekamer G.E. & Reijns G.L., *Computer Performance*, (L96), Computer Architecture Laboratory, Faculty of Electrical Engineering, Delft University of Technology, 1986

Howard R.E., Jackel L.D. & Graf H.P., *Electronic neural networks*, AT&T Technical Journal, Volume 67, Issue 1, January / February 1988

Gao Q. & Liu Z., *Spatio-temporal associative memory and a high-order correlation neural network*, Institute of Biophysics, Academia Sinica, Beijing, Peoples Republic of China. Published in INNS, "Abstracts of the first annual INNS meeting, Boston, 1988", Neural Networks, Volume 1, Supplement 1, pp. 176, Pergamon Press, 1988

INNS, *Abstracts of the first annual INNS meeting, Boston, 1988*, Neural Networks, Volume 1, Supplement 1, Pergamon Press, 1988

Kaski K. & Vanhala J., *Simulating neural networks in a distributed environment*, Tampere University of technology, Microelectronics laboratory, P.O. Box 527, SF-33101 Tampere, Finland. Published in the proceedings of the 11th Occam User Group technical meeting, Edinburgh, Scotland, September 1989

Rosenberg C.R. & Sejnowski T., *NETtalk: A Parallel Network That Learns to Read Aloud*, John Hopkins Univ., Technical Report JHU/EECS-86/01, 1986

Soucek B. & Soucek M., *Neural and Massively Parallel Computers*, John Wiley & Sons, Inc, 1988

VISUALIZATION OF EVENTS IN A SPR NEURAL NETWORK

The SPR neural network for garbled text string recognition has been implemented as a software simulation program called SPR. The SPR program offers the possibility to **visualize** the time varying activation levels of the PE's in the network while an input name is being processed. There are two different visualization methods: the Cathode Ray Tube **(CRT) display** method and the **filed display** method. Both methods will be explained.

A.1 Description of the architecture of the SPR network

To interpret the visualization it is necessary to understand the SPR neural network architecture. A short explanation of the SPR architecture is given in this appendix. A detailed description can be found in the report (chapter 3).

The input to the network is divided into **time slices**. During each time slice, one character of the input word is processed by the network. The input name 'URK' would be divided into three time slices: Slice 1 = 'U'
Slice 2 = 'R'
Slice 3 = 'K'

First all PE's of the network will process input character 'U' corresponding time slice 1. Next all PE's will process input character 'R' corresponding to time slice 2. Finally all PE's will process input character 'K' corresponding to time slice 3.

Each PE represents one character. The PE's of the network are arranged as a matrix. Each **row** represents one input name. Each **column** represents one time slice. The first column represents time slice 1, the second column represents time slice 2, etc. The names are right aligned, i.e. if a name contains N (N<C) characters and there are C columns in the matrix, then the leftmost (C-N) PE's are **not used**. There are feed forward (left to right) connections between PE's belonging to the same row

A.2 Representation of the activation level of a PE

Each PE is represented by a solid horizontal bar. The length of the bar represents the **activation level** of the PE (a longer bar means a higher activation level). The activation levels are scaled.

The highest PE activation level that ever occurs while processing an input name is used as a reference activation level of 100%. This 100% activation level is assigned to the maximum bar length. When, in a certain time slice, a PE has an activation level of 40% of the maximum activation level, the length of the bar representing that PE will be set to 40% of the maximum bar length.

The activation level of a PE can optionally be displayed numerically (as a floating point number) instead of graphically (as a bar). The advantage of the numerical display mode is that the PE activation levels are known precisely. There is only a limited choice of bar lengths to choose from in the graphical display mode, so activation levels that differ only slightly may be visualized as having the same bar length and thus having the same activation level. The numerical display mode will remove this uncertainty. The disadvantage of the numerical display mode is that it is more difficult to see at once what happens in the network.

A.3 CRT display method

The CRT display method shows the **most significant subset** of the events in the network as a plot on the CRT. The advantage of this method is that the reaction of the network to an input name can be seen **at once** in a single picture. The disadvantage is that only a subset of the events is shown. In almost *all cases* this display method is sufficient to analyse the reaction of the network.

Suppose an input name contains M characters and the SPR network consists of a matrix of M columns and L rows (i.e. there are L names known to the network). These M characters are processed by the network in M time slices. The CRT display method displays a matrix of M columns and L rows. The leftmost column (column 1) shows the activation levels of the PE's in the leftmost column of the SPR network matrix at time slice 1 (when the first character is processed). The next column (column 2) shows the activation levels of the PE's in the second column of the SPR network matrix at time slice 2 (when the second character is processed), etc. The rightmost column (column M) shows the activation levels of the PE's in the last column of the SPR network matrix at time slice M (when the last character is processed).

Suppose that the SPR network consists of C columns and L rows where C>M. Due to the limited space available on a single screen, the CRT display method only shows M (M=8) of the C time slices (columns), i.e. slices C-M+1 to C. This means that when a name containing N_1 (N_1>M)

characters is processed, only the M time slices starting with slice (C-M+1) are shown. When a name containing N_2 (N_2<M) characters is processed, the leftmost (M-N_2) columns are blank.

Usually **one PE** in the last column has the **highest** activation value. The row to which this PE belongs is the winning row (layer) of the network. So the winning name (i.e. the name that corresponds to this row) is known. There are some exceptions to this rule. These exceptions are explained in the report (sections 3.3 to 3.6).

A.4 The filed display method

The filed display method presents **all events** that occurred while an input name was processed. For each time slice there is a plot of the activation levels of all PE's in the network. This allows a comprehensive analysis of the reaction of the network to an input name. The data is stored as a file. This file can be shown on a CRT or it can be printed on any printer that supports the IBM extended character set. At the start of the file there is a list of the current parameter settings of the network, the input name, the resulting output name, and the name templates. Following this information there is one plot for each time slice.

A.5 Examples of the CRT display method

This series of four examples consists of copies of the result of the CRT display method. Each copy shows the reaction of the network to a particular input name. The network consists of 12 columns (i.e. the maximum length of a name is 12 characters) and 11 rows (i.e. the network can recognize 11 different names). The first input name 'ALKMAAR' (figure 1.a) perfectly matches with the first row (layer) of the network, which represents the name 'ALKMAAR'. The result of this perfect match is a maximum activation level of the last (rightmost) PE of the first row.

The next input names 'ARKMAAR', 'ARKMAAN' and 'RKMAAN' (figures 1.b, 1.c and 1.d) show what happens when there is a decreasing match between the input name and the names known to the network. The network still recognizes the input name correctly, but the difference between the winning layer and the other layers in the network decreases.

A.6 Examples of the filed display method

Two examples of the filed display method are presented. The first example (figures 2.a to 2.i) is a copy of the contents of the output file that is produced when the perfectly matching name 'ALKMAAR' is processed. The second example (figures 3.a to 3.h) is produced when the badly garbled name 'RKMAAN' is processed.

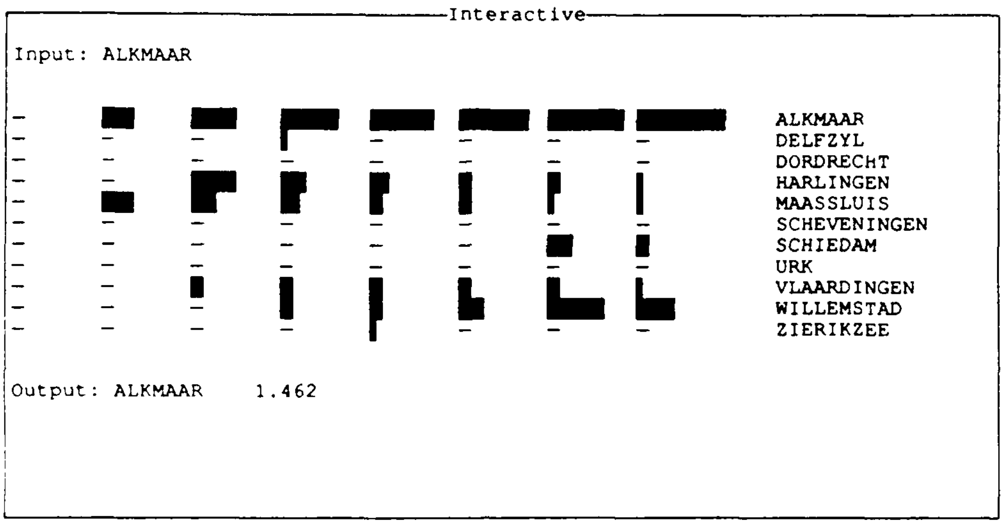

Figure 1.a CRT display method, input name is 'ALKMAAR'

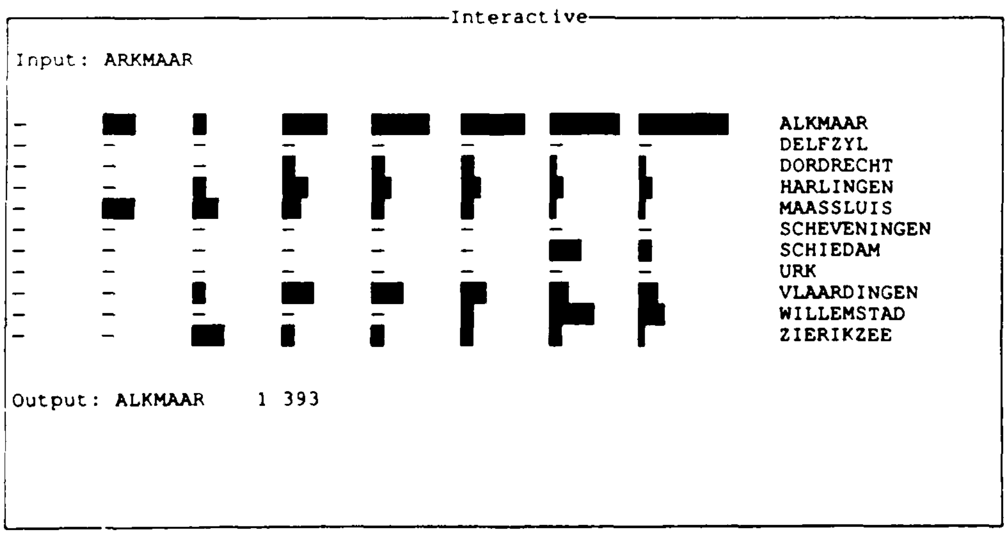

Figure 1.b CRT display method, input name is 'ARKMAAR'

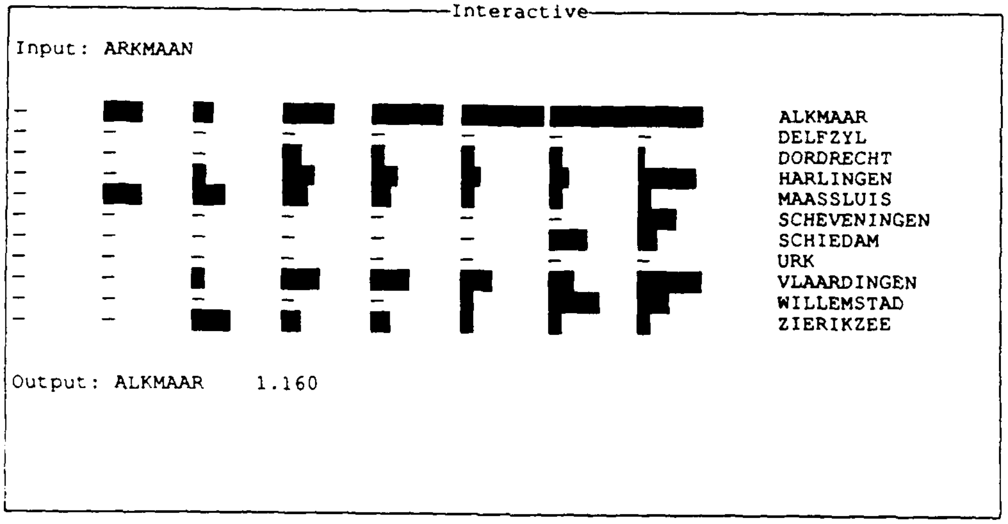

Figure 1.c CRT display method, input name is 'ARKMAAN'

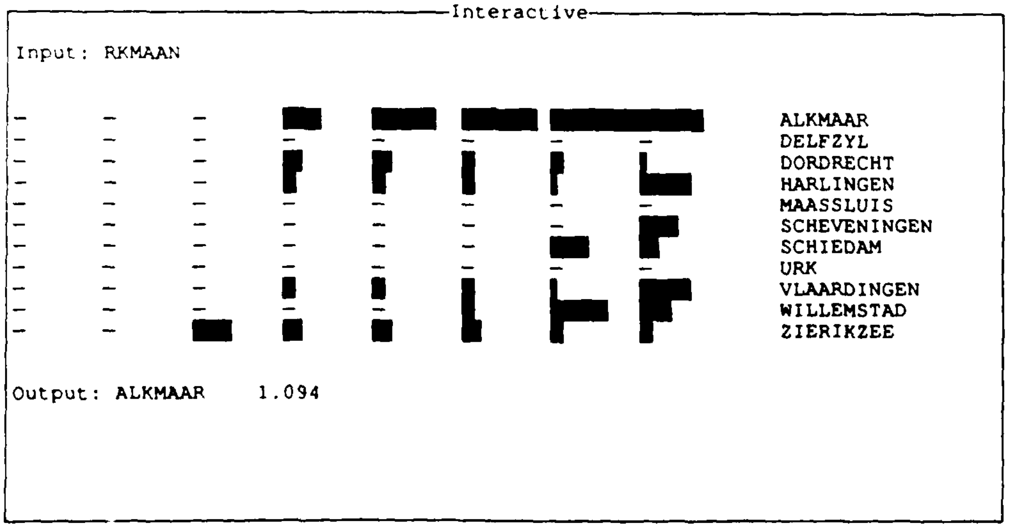

Figure 1.d CRT display method, input name is 'RKMAAN'

```
 ----------------------------------------------
| String Recognition with an SPR Neural Network |
 ----------------------------------------------


Parameters:

  0.800 Dc      Decay Constant
  0.400 Tsg1    first order Time Sequence Gain
  0.300 Tsg2    second order Time Sequence Gain
  1.000 Ig      Input Gain
  0.500 Thr     activation Threshold
  1.300 Av      Attack Value

Input  : ALKMAAR

Output : ALKMAAR
```

Figure 2.a Filed display method, input name is 'ALKMAAR', parameters

```
Name templates:

.  .  .  .  .  A  L  K  M  A  A  R
.  .  .  .  .  D  E  L  F  Z  Y  L
.  .  .  D  O  R  D  R  E  C  H  T
.  .  .  H  A  R  L  I  N  G  E  N
.  .  .  M  A  A  S  S  L  U  I  S
S  C  H  E  V  E  N  I  N  G  E  N
.  .  .  .  S  C  H  I  E  D  A  M
.  .  .  .  .  .  .  .  .  U  R  K
.  V  L  A  A  R  D  I  N  G  E  N
.  .  W  I  L  L  E  M  S  T  A  D
.  .  .  Z  I  E  R  I  K  Z  E  E
```

Figure 2.b Filed display method, input name is 'ALKMAAR', name templates

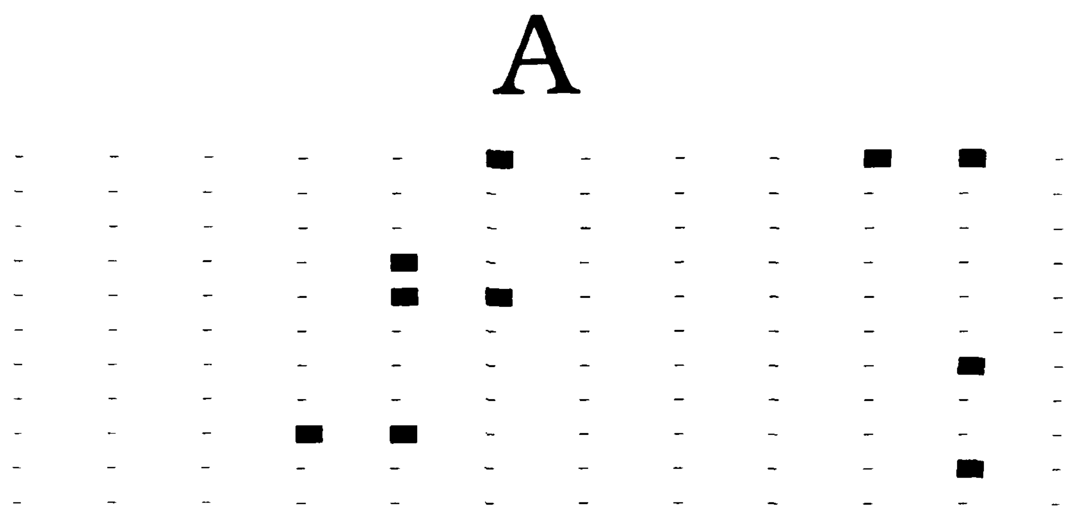

Figure 2.c Filed display method, input name is 'ALKMAAR', input character is 'A'

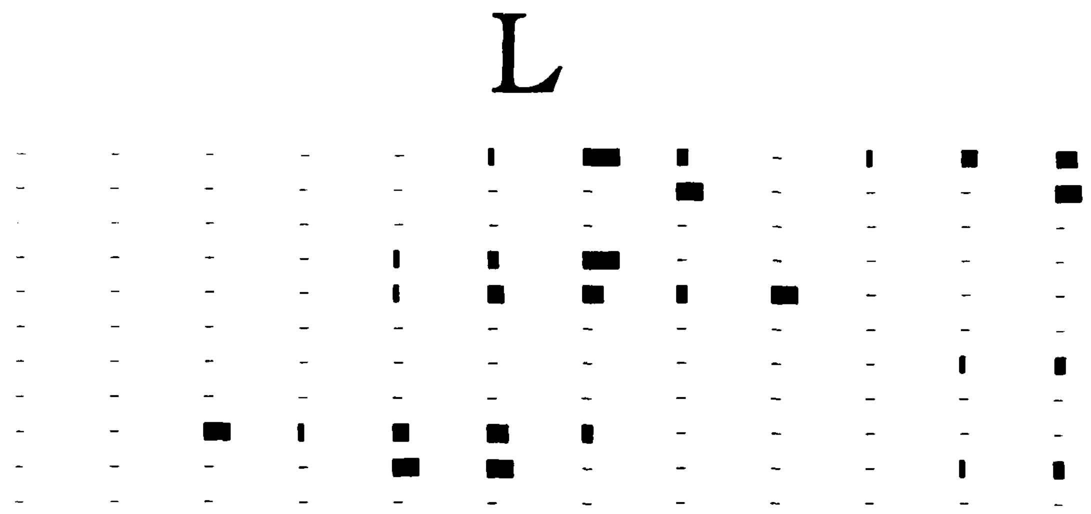

Figure 2.d Filed display method, input name is 'ALKMAAR', input character is 'L'

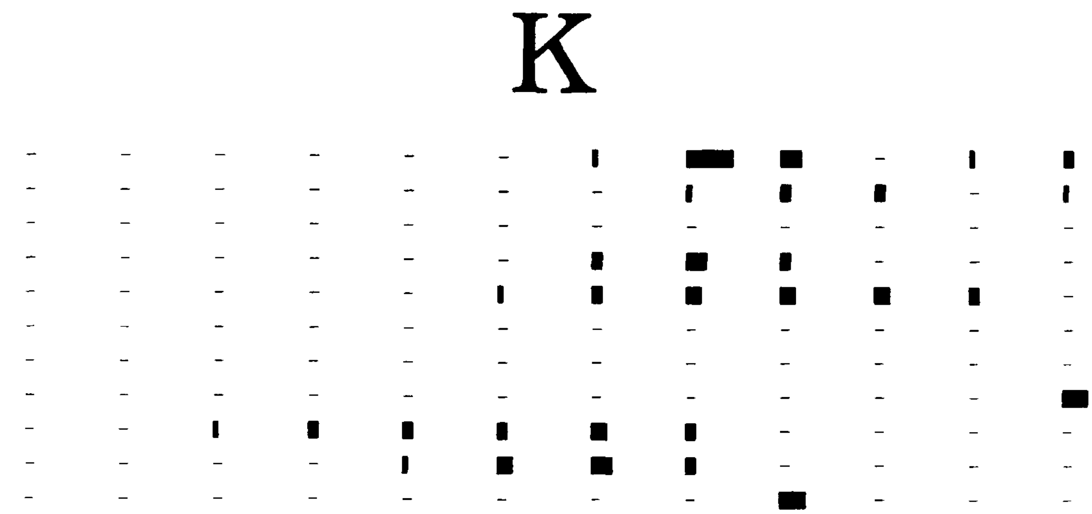

Figure 2.e Filed display method, input name is 'ALKMAAR', input character is 'K'

Figure 2.f Filed display method, input name is 'ALKMAAR', input character is 'M'

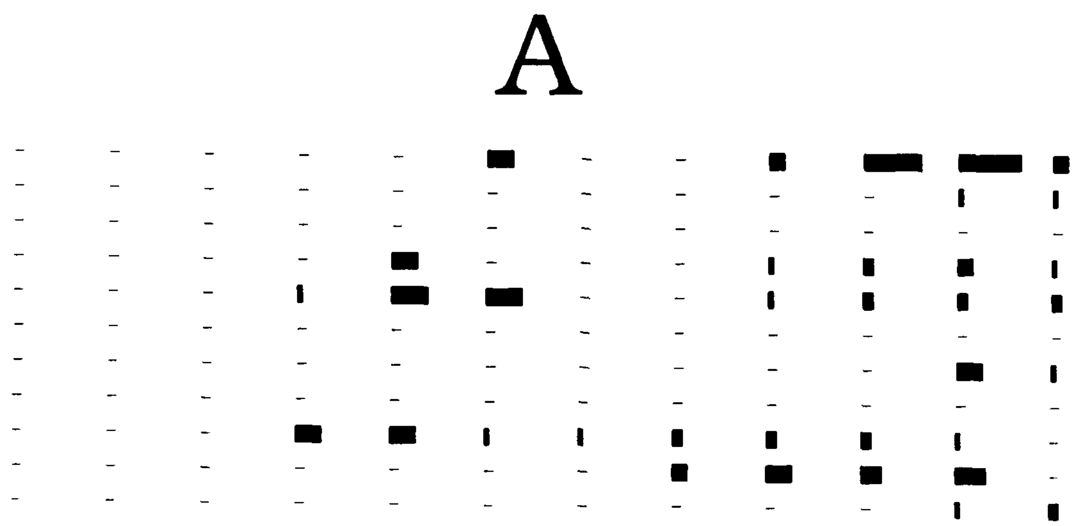

Figure 2.g Filed display method, input name is 'ALKMAAR', input character is 'A'

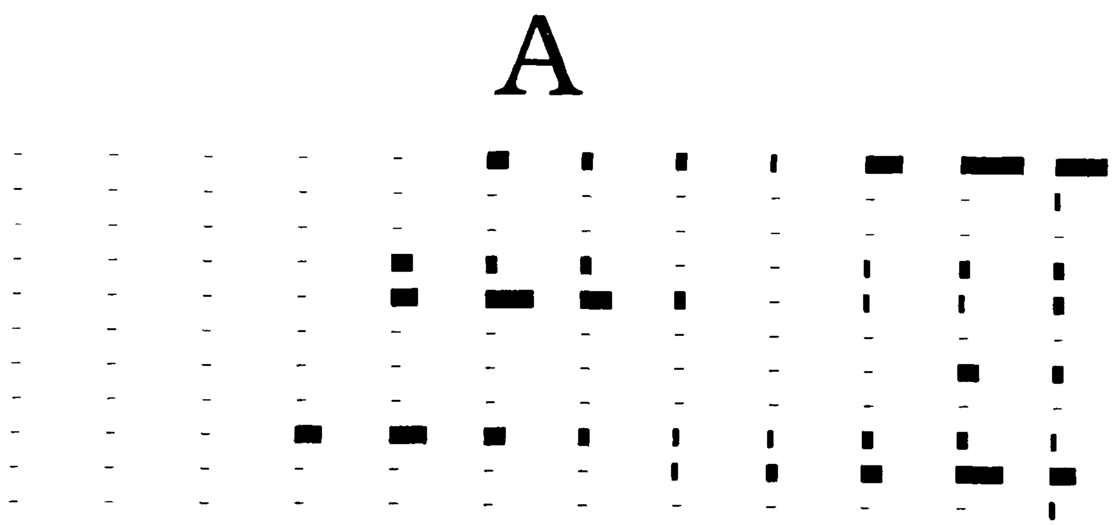

Figure 2.h Filed display method, input name is 'ALKMAAR', input character is 'A'

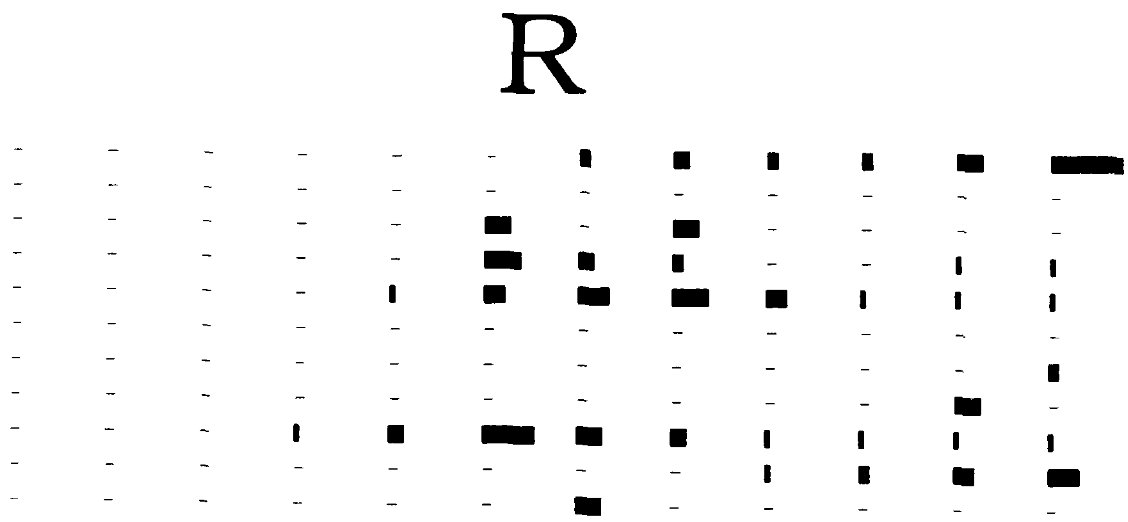

Figure 2.i Filed display method, input name is 'ALKMAAR', input character is 'R'

```
---------------------------------------------
| String Recognition with an SPR Neural Network |
---------------------------------------------


 Parameters:

   0.800 Dc      Decay Constant
   0.400 Tsg1    first order Time Sequence Gain
   0.300 Tsg2    second order Time Sequence Gain
   1.000 Ig      Input Gain
   0.500 Thr     activation Threshold
   1.300 Av      Attack Value

 Input  : RKMAAN

 Output: ALKMAAR
```

Figure 3.a Filed display method, input name is 'RKMAAN', parameters

```
Name templates:
.    .    .    .    .    A    L    K    M    A    A    R
.    .    .    .    .    D    E    L    F    Z    Y    L
.    .    .    D    O    R    D    R    E    C    H    T
.    .    .    H    A    R    L    I    N    G    E    N
.    .    .    M    A    A    S    S    L    U    I    S
S    C    H    E    V    E    N    I    N    G    E    N
.    .    .    .    S    C    H    I    E    D    A    M
.    .    .    .    .    .    .    .    ..   U    R    K
.    V    L    A    A    R    D    I    N    G    E    N
.    .    W    I    L    L    E    M    S    T    A    D
.    .    .    Z    I    E    R    I    K    Z    E    E
```

Figure 3.b Filed display method, input name is 'RKMAAN', name templates

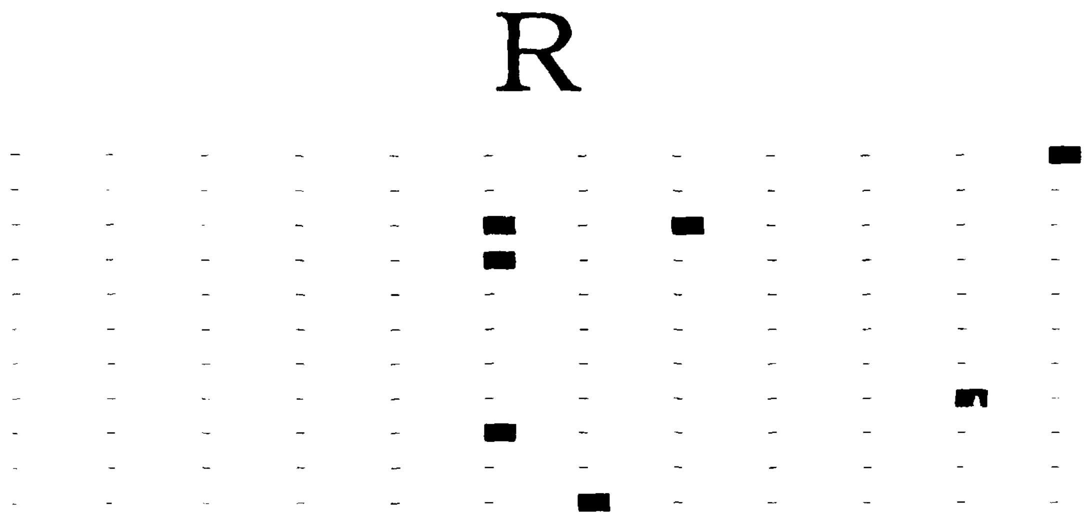

Figure 3.c Filed display method, input name is 'RKMAAN', input character is 'R'

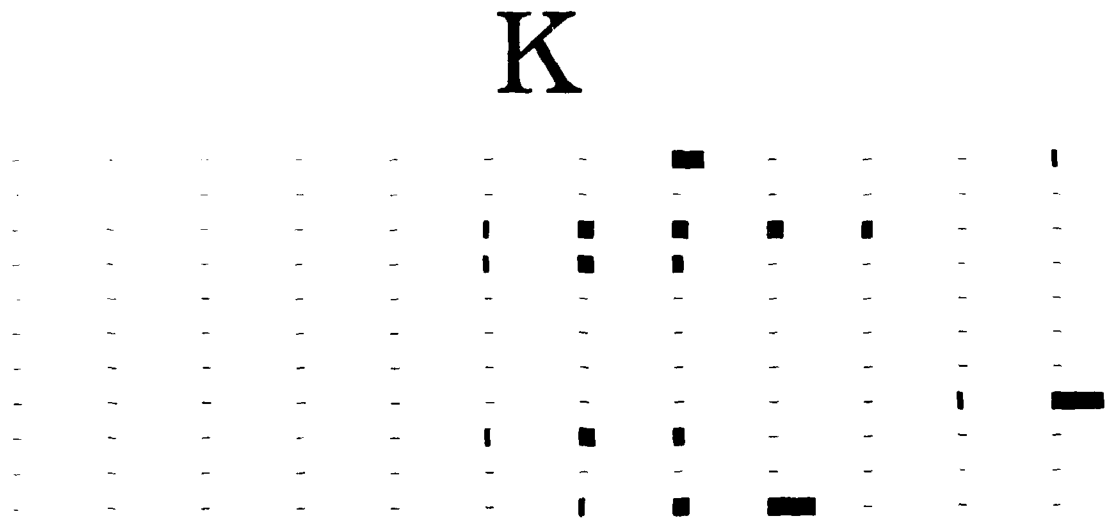

Figure 3.d Filed display method, input name is 'RKMAAN', input character is 'K'

Figure 3.e Filed display method, input name is 'RKMAAN', input character is 'M'

Figure 3.f Filed display method, input name is 'RKMAAN', input character is 'A'

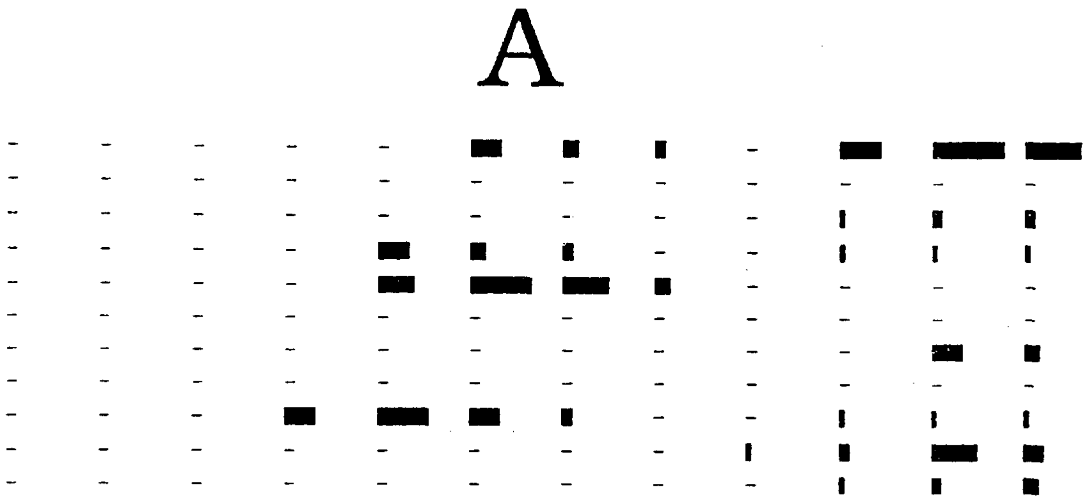

Figure 3.g Filed display method, input name is 'RKMAAN', input character is 'A'

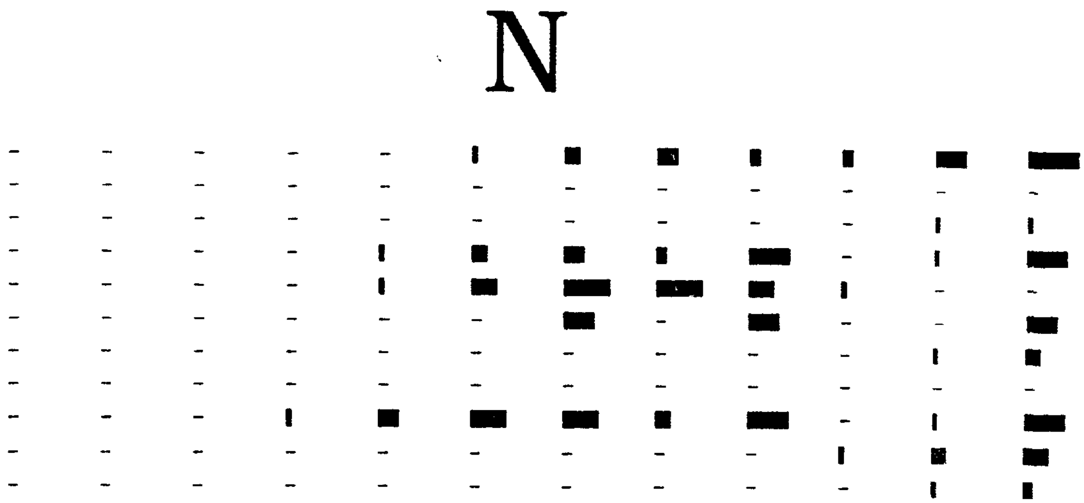

Figure 3.h Filed display method, input name is 'RKMAAN', input character is 'N'

REPORT DOCUMENTATION PAGE (MOD-NL)

1. DEFENSE REPORT NUMBER (MOD-NL)
TD90-1716

2. RECIPIENT'S ACCESSION NUMBER

3. PERFORMING ORGANIZATION REPORT
FEL-90-B131

4. PROJECT/TASK/WORK UNIT NO
20477

5. CONTRACT NUMBER

6. REPORT DATE
JUNE 1990

7. NUMBER OF PAGES
67 (incl. titlepage and appendix, excl. distr. list and RDP)

8. NUMBER OF REFERENCES
13

9. TYPE OF REPORT AND DATES
FINAL REPORT

10. TITLE AND SUBTITLE
GARBLED TEXT STRING RECOGNITION WITH A SPATIO-TEMPORAL PATTERN RECOGNITION NEURAL NETWORK

11. AUTHOR(S)
P.P. MEILER

12. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

TNO PHYSICS AND ELECTRONICS LABORATORY
P.O. BOX 96864 2509 JG THE HAGUE THE NETHERLANDS

13. SPONSORING/MONITORING AGENCY NAME(S)

14. SUPPLEMENTARY NOTES

PAPER PRESENTED AT:
- NEURO-NIMES'89 INTERNATIONAL WORKSHOP, NIMES, FRANCE, NOVEMBER 13-16, 1989
- AI APPLICATIONS'89, SECOND DUTCH CONFERENCE, THE HAGUE, THE NETHERLANDS, NOVEMBER 28-29, 1989

15. ABSTRACT

THE PURPOSE OF THIS PROJECT IS TO SHOW THAT NEURAL NETWORKS CAN BE USED TO RECOGNIZE GARBLED WORDS AND/OR PARTS OF SENTENCES IN A REAL-WORLD APPLICATION. TNO-FEL HAS STUDIED AND BUILT AN APPLICATION THAT RECOGNIZES THE NAMES OF SHIPS. THESE NAMES MAY BE GARBLED BY TRANSMISSION OR TYPING ERRORS, AND SYNONYMS OR CORRUPTIONS MAY BE USED. THE SPR NETWORK EMPHASIZES THE CHARACTER-SEQUENCE RELATIONSHIPS WITHIN WORDS. SPR IS PROOF AGAINST MISSING, EXTRA OR INTERCHANGED (PAIRS OF) CHARACTERS. A LEARNING STRATEGY WAS DEVELOPED AND IMPLEMENTED. SEVERAL MEASUREMENTS WERE PERFORMED.

16. DESCRIPTORS

NEURAL NETWORKS
PATTERN RECOGNITION
CHARACTER RECOGNITION
PARALLEL PROCESSING
REAL TIME COMPUTATION

IDENTIFIERS

TEXT STRING RECOGNITION
SPATIO-TEMPORAL TECHNIQUES

17a. SECURITY CLASSIFICATION (OF REPORT)
UNCLASSIFIED

17b. SECURITY CLASSIFICATION (OF PAGE)
UNCLASSIFIED

17C. SECURITY CLASSIFICATION (OF ABSTRACT)
UNCLASSIFIED

18. DISTRIBUTION/AVAILABILITY STATEMENT
UNLIMITED

17d. SECURITY CLASSIFICATION (OF TITLES)
UNCLASSIFIED